

© fotolia 62904980

# Web Application Firewall Bypassing



# Bypassing a WAF – Why?

---

- Number of deployed Web Application Firewalls (WAFs) is increasing
- WAFs make a penetration test more difficult
- Attempting to bypass a WAF is an important aspect of a penetration test

# Main Goal

---

- Understand the limits of WAFs
- Provide a practical approach to bypass WAFs for security experts in order to ensure accurate assessment results

# Introduction to Web Application Firewalls

# Overview

---

- Replaces old fashioned Firewalls and IDS/IPS
- Understands HTTP traffic better than traditional firewalls
- Protects a web application by adding a security layer
- Checks for malicious traffic and blocks it

# Overview

# WEB APPLICATION FIREWALL



<https://www.e2enetworks.com/help/glossary/waf-web-application-firewall/>

# Functionality



- Pre-processor:

Decide whether a request will be processed further

- Normalization:

Standardize user input

- Validate Input:

Check user input against rules

# Normalization Functions

- Simplifies the writing of rules
- No Knowledge about different forms of input needed

Function Name	Description
compressWhitespace	converts whitespace chars to spaces
hexDecode	decodes a hex-encoded string
lowercase	converts characters to lowercase
urlDecode	decodes a URL-encoded string



# Input Validation

---

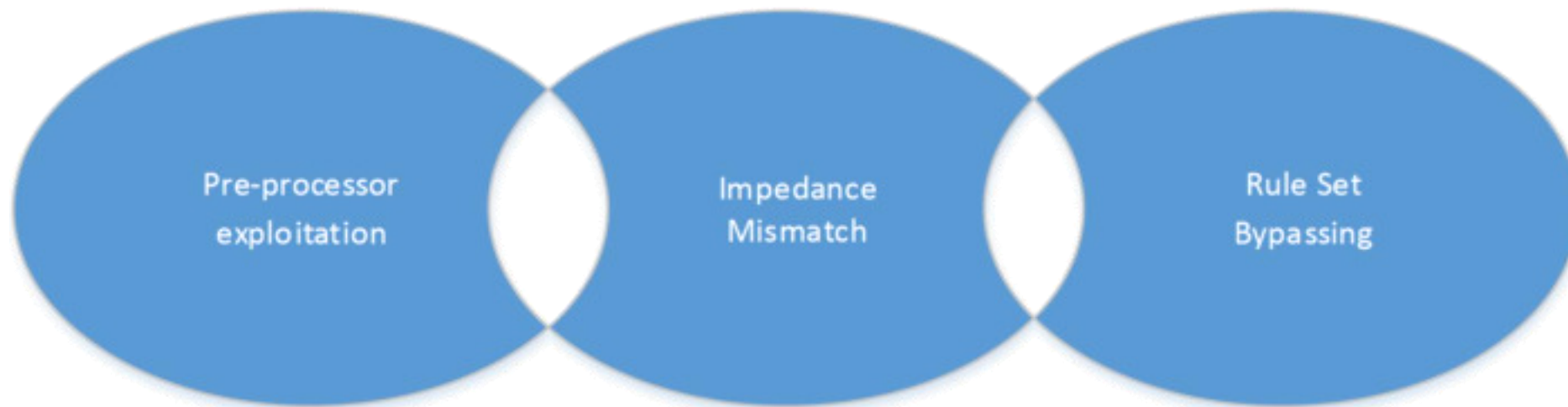
- Security Models define how to enforce rules
- Rules consist of regular expressions
- Three Security Models:
  - Positive Security Model
  - Negative Security Model
  - Hybrid Security Model

# Security Models

<b>Positive Security Model (Whitelist)</b>	<b>Negative Security Model (Blacklist)</b>
Deny all but known good	Allow all but known bad
Prevents Zero-day Exploits	Shipped with WAF
More secure than blacklist	Fast adoption
Comprehensive understanding of application is needed	Little knowledge needed
Creating rules is a time-consuming process	Protect several applications
	Tends to false positives
	Resource-consuming

# Bypassing Methods and Techniques

# Overview



## **Pre-processor Exploitation:**

Make WAF skip input validation

## **Impedance Mismatch:**

WAF interprets input differently than back end

## **Rule Set Bypassing:**

Use Payloads that are not detected by the WAF

# Pre-processor Exploitation

# Skipping Parameter Verification

- PHP removes whitespaces from parameter names or transforms them into underscores

```
http://www.website.com/products.php?%20productid=select 1,2,3
```

- ASP removes % character that is not followed by two hexadecimal digits

```
http://www.website.com/products.aspx?%productid=select 1,2,3
```

- A WAF which does not reject unknown parameters may be bypassed

# Malformed HTTP Methods

- Misconfigured web servers may accept malformed HTTP methods

The screenshot displays two panels: 'Request' and 'Response'. The 'Request' panel has tabs for 'Raw', 'Params', 'Headers', and 'Hex'. The 'Raw' tab is selected, showing the text 'HELLO123 /zielgruppen/studier' and 'Host: frankfurt-university.de'. The 'HELLO123' part is highlighted with a red box. The 'Response' panel has tabs for 'Raw', 'Headers', 'Hex', 'HTML', and 'Render'. The 'Raw' tab is selected, showing the text 'HTTP/1.0 200 OK' and 'Date: Thu, 03 Sep 2015 12:27:23 GMT'. The 'HTTP/1.0 200 OK' part is highlighted with a red box.

- A WAF that only inspects GET and POST requests may be bypassed

# Overloading the WAF

---

- A WAF may be configured to skip input validation if performance load is heavy
- Often applies to embedded WAFs
- Great deal of malicious requests can be sent with the chance that the WAF will overload and let some requests through



# Impedance Mismatch

# HTTP Parameter Pollution

- Sending a number of parameters with the same name
- Technologies interpret this request

```
http://www.website.com/products/?productid=1&productid=2
```

differently:

Back end	Behavior	Processed
ASP.NET	Concatenate with comma	productid=1,2
JSP	First Occurrence	productid=1
PHP	Last Occurrence	productid=2

# HTTP Parameter Pollution

- The following payload

```
productid=select 1,2,3 from table
```

- can be divided:

```
?productid=select 1&productid=2,3 from table
```

- WAF sees two individual parameters and analyzes both values independently
- ASP.NET back end concatenates both values

# Double URL Encoding

- WAF normalizes URL encoded characters into ASCII text
- The WAF may be configured to decode characters only once
- Double URL Encoding a payload may result in a bypass
- The following payload contains a double URL encoded character

**'s' -> %73 -> %25%37%33**

**1 union %25%37%33elect 1,2,3**

# Rule Set Bypassing

# Bypass Rule Set

---

- Two methods:
  - Brute force by enumerating different payloads
  - Reverse-engineer the WAFs rule set

# Approach for Penetration Testers

# Overview

---

- Similar to the phases of a penetration test
- Divided into six phases, whereas Phase 0 may not always be possible



*Objective:* **find security flaws in the application more easily**

- Allows a more focused approach when the WAF is enabled again
- More accurate results for the customer
- May not be realizable in some penetration tests

# Phase 1 - Reconnaissance

---

*Objective:* **gather information to get an overview of the target**

- Basis for the subsequent phases
- Gather information about:
  - web server
  - programming language
  - WAF & Security Model
  - Internal IP Addresses

## Phase 2 – Attacking The Pre-processor

---

### *Objective:* make the WAF skip input validation

- Identify which parts of a HTTP request are inspected by the WAF to develop an exploit:
  1. Send individual requests that differ in the location of a payload
  2. Observe which requests are blocked
  3. Attempt to develop an exploit

## Phase 3 – Finding an Impedance Mismatch

---

***Objective:* make the WAF interpret a request differently than the back end and therefore not blocking it**

- Knowledge about back end technologies is needed

## Phase 5 – Finding Other Vulnerabilities

---

***Objective:* find other vulnerabilities that can not be detected by the WAF**

- Broken authentication mechanism
- Privilege escalation vulnerabilities
- etc.

### *Objective:* report vulnerabilities to customer

- Advise customer to fix the **root cause** of a vulnerability
- For the time being, the vulnerability should be **virtually patched** by adding specific rules to the WAF
- Explain that the WAF can help to mitigate a vulnerability, but is **not a water-proof solution**

# WAFNinja

# Overview

---

- CLI Tool written in Python
- Automates parts of the approach
- Used by security experts world-wide 😊
- Supports
  - GET and POST parameter
  - Usage of cookies
  - Usage of an intercepting proxy



# Overview

- Published on GitHub 😊

The screenshot shows the GitHub repository page for 'khalilbijjou / WAFNinja'. At the top, it displays the repository name and navigation options: Watch (35), Star (442), and Fork (163). Below this, there are tabs for Code, Issues (3), Pull requests (0), Projects (0), and Insights. A description states: 'WAFNinja is a tool which contains two functions to attack Web Application Firewalls.' Below the description, it shows repository statistics: 25 commits, 1 branch, 0 releases, and 4 contributors. There are buttons for 'Branch: master', 'New pull request', 'Find File', and 'Clone or download'. A commit history table is visible below, showing the latest commit by khalilbijjou updating README.md on 6 Dec 2017. The table lists several files and their commit messages:

File	Commit Message	Time
db	Delete __init__.pyc	3 years ago
ninja	added install notes and quick dirty fix for XSS issue in output repor...	2 years ago
README.md	Update README.md	2 years ago
argument.py	Add files via upload	3 years ago
requirements.txt	Create requirements.txt	3 years ago
wafninja.py	Add files via upload	3 years ago

- <https://github.com/khalilbijjou/WAFNinja>

# Fuzzing

---

- Sends different symbols and keywords
- Analyzes the response
- Results are displayed in a clear and concise way
- Fuzzing strings can be
  - shared within a team
  - extended with the insert-fuzz function

# Demo

# Never forget!

---

Always fix the **root cause** of your vulnerabilities