****Софийски Университет**

**Проект на тема**

**Атаки с двойно подадени параметри при обратни прокси сървъри**

**За дисциплината**

**Основи на сигурното уеб програмиране**

**Съдържание**

**1) Въведение 2**

**2)** **Запознаване с обратните прокси сървъри 3**

**3)** **HPP , и как различните сървъри обработват дублиращи се параметри 5**

**4)** **Уязвимостта 8**

**5) Предпазване от атаката 10**

**6) Източници 12**

**Въведение**

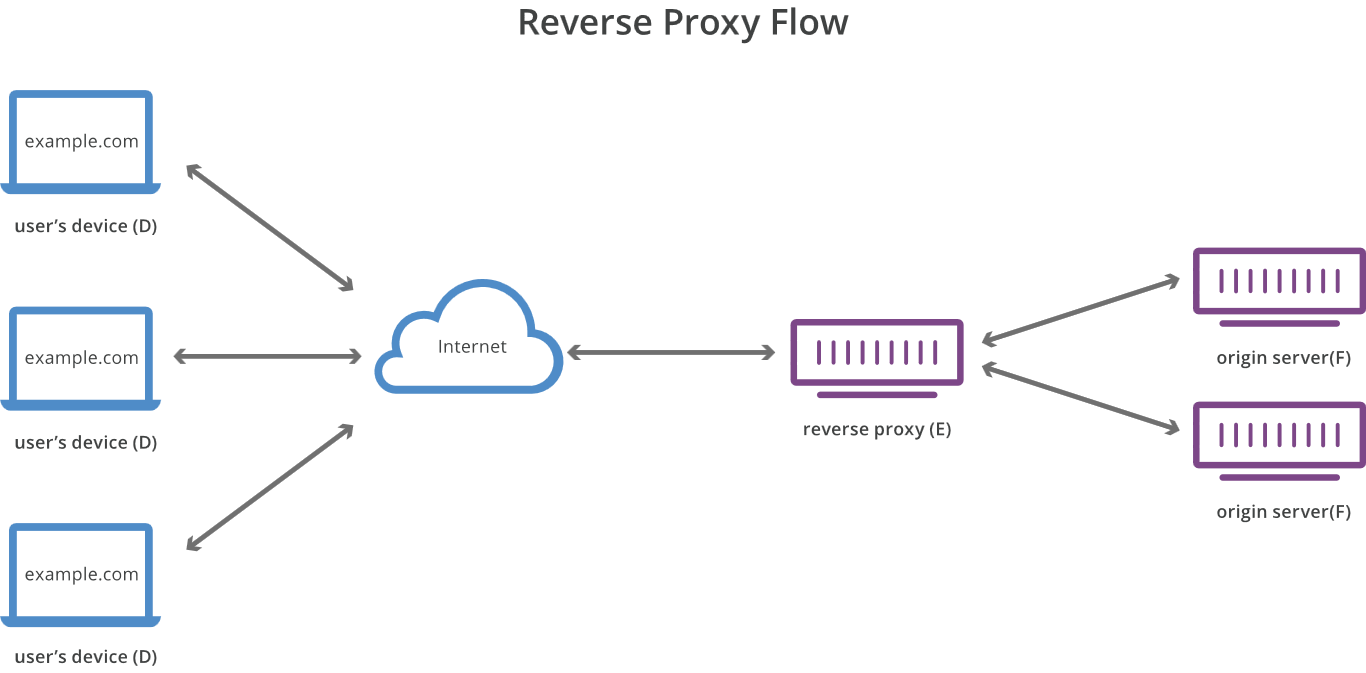
В настоящия доклад се разглежда темата „Атаки с двойно подадени параметри при обратни прокси сървъри“. Както подсказва името и, атаката се състои в подаване на две стойности на 1 GET параметър. Тя се възползва от факта, че различните типове сървъри могат да разчетат различни стойности за тази променлива.Така тя може да подаде потенциално опасни данни на backend сървъра, дори когато reverse proxy трябва да защитава от подобна атака.

Ще се наблегне на функционалността на обратните прокси сървъри, в кои случай те са податливи на тази атака, и евентуалните последствия от нея , придружени с няколко примера.

И естествено ще се покажат начините за предпазване от атаката.

**Запознаване с обратните прокси сървъри**

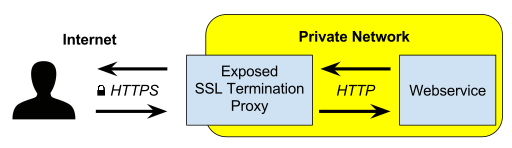
Прокси сървърът е сървърно приложение или устройство, което действа като посредник при заявки от клиенти, търсещи ресурси от други сървъри, предоставящи тези ресурси.

Обратният прокси е характерен с това, че седи пред backend сървърите на едно уеб приложение, и може да препраща клиентските заявки към тези сървъри. След това ресурсите, получени от вътрешните сървъри, се връщат на клиента така, сякаш произхождат от самия прокси сървър.

Така reverse proxy може да помогне за повишаване на сигурността, производителността и надеждността.

Употребите му биват:

**1. Load balancing** – когато функционалността на един сайт се поддържа от няколко сървъра, reverse proxy може да подсигури равномерното разпределение на потребителските заявки между тях. Така се избягва претоварване на отделен сървър, а ако се наложи да бъде изваден от употреба някой сървър,или да се пусне в употреба нов, чрез пренасочване на трафика сайтът може да продължи работа.

**2. Криптиране** – TLS/ SSL криптиране и декриптиране са изчислително скъпи операции.Поради това е от полза те да се извършват само от прокси сървъра. Така потребителя и проксито си общуват чрез HTTPS, а до backend сървърите информацията достига некриптирана (връзката proxy <-> backend се счита за сигурна ).

**3.Кеширане** – за някои типове потребителски заявки, проксито може временно да запази получените от вътрешните сървъри отговори. Така , ако следващите постъпили заявки са за същото, те ще бъдат обслужени от данните в кеша на проксито. По този начин времето за отговор и натоварването на backend-а значително могат да се намалят.

**4.Защита от атаки** – Когато сайт има reverse proxy, адресите на backend сървърите остават скити за потребителите. Нападателите ще могат да се насочват само към прокси сървъра, като така DoS атаките стават по-трудни. Поради същата причина reverse proxy е ключовото място за firewalls - за филтриране на нежеланото съдържание. Добре защитеното прокси става предпазна стена за системата на сайта. Но при прекалена вяра в неговите възможности и оставяне на уязвимости в задните сървъри , сайтът рискува да стане жертва на друг тип атака.

**HPP , и как различните сървъри обработват дублиращи се параметри**

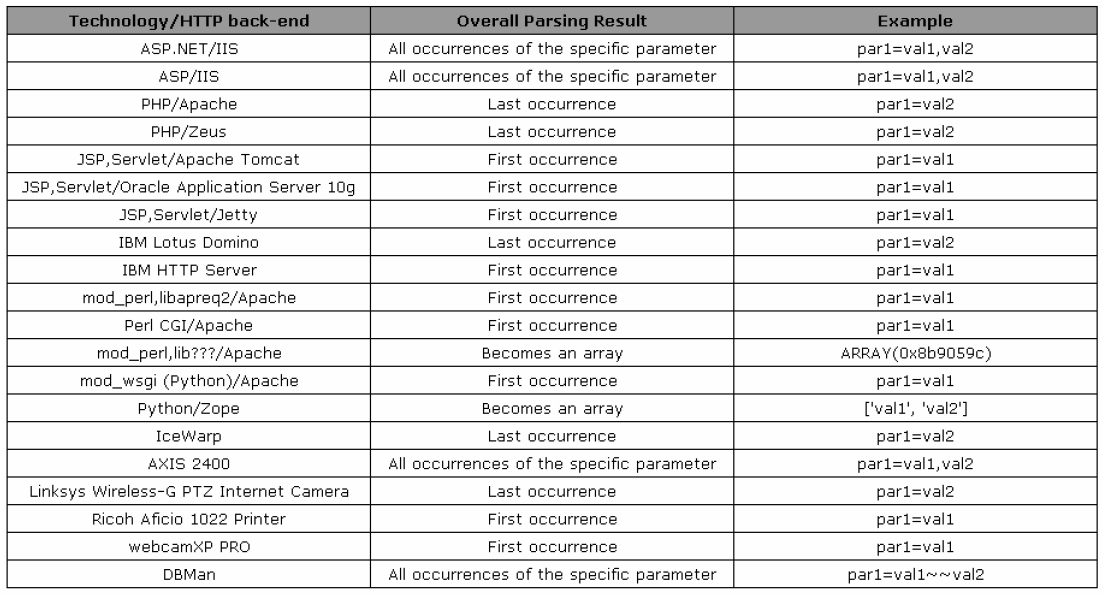
Когато става въпрос за reverse proxy , от особено значение са познанията за атаката HTTP Parameter Pollution . HPP възниква, когато система приема множество параметри с едно и също име и ги обработва по начин, който може да е несигурен или неочакван. Така злонамерено лице може чрез параметрите да вмъкне свои параметри в сървъра или страницата, ако те преди това не преминат през процес на „Sanitization“.

Въздействието може да варира: промяна на съществуващите твърдо кодирани HTTP параметри, промяна на поведението на приложение, да даде на атакуващия достъп до скрити данни, да заобикаля firewalls. Така се отваря възможността за много други, още по-сериозни атаки към уеб приложението.

На пръв поглед HPP не изглежда толкова сериозна. Чрез предварително филтриране на прочетените стойности , атаката ще бъде опропастена. Но с намесата на reverse proxy проблемът става по-сложен.

Няма точно определен начин, по който сървърите да обработят url параметър с няколко подадени стойности. Един тип сървъри могат да вземат само последната стойност, други – само първата, трети – всички.

Таблицата е взета от : [[2]](#и2)



За по-нагледно,ще бъдат разгледани методите за взимане на url параметър в самите сървъри.

**PHP/Apache** [[5]](#и5)

$\_GET - Асоциативен масив от променливите, получени от URL.

$\_GET["name"] – връща стойността на променливата name, като ако тя има няколко стойности, ще бъде получена само последната .

Пр: http://example.com/?name=alice&name=bob

echo $\_GET[ "name" ] ; - връща „bob“

**JSP/ Apache Tomcat** [[6]](#и6)

Върху обект от тип HttpServletRequest, чрез прилагане на getParameter(java.lang.String name) се получава стойността на исканият параметър като String или null, ако параметърът не съществува.

Ако този метод бъде използван с параметър с няколко стойности, върнатата стойност е равна на първата. В този случай е по-добре да се ползва getParameterValues, връщащ масив от всичките стойности.

Пр: http://example.com/?name=alice&name=bob

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

String getname= request.getParameter("name");

}

Така в getname се получава стойността „alice“ .

**Python/Apache** [[7]](#и7)

В библиотеката urllib.parse

urlparse (url) връща named tuple на различните компоненти на url-a.

Оттам parse\_qs(qs) върху компонента отговарящ за заявката ще върне речник, с ключове имената на параметрите, и стойности – листове от подадените им в url-a стойности.

Пр : import urllib.parse as urlparse

from urllib.parse import parse\_qs

url = 'http://example.com/?name=alice&name=bob'

parsed = urlparse.urlparse(url)

print(parse\_qs(parsed.query)['name'])

ще върне - ['alice' , 'bob']

**Perl(CGI)/Apache** [[8]](#и8)

В библиотеката CGI на Perl

$query->param('foo') ще извлече първата стойност на посочения параметър.

Ако параметърът може да има няколко стойности е препоръчително да се ползва

$query->multi\_param('foo') - така се взима списък с всичките стойности.

Пр: http://example.com/?name=alice&name=bob

use CGI;

my $q = CGI->new;

my $value = $q->param(' name ');

В променливата се запазва alice.

**ASP/IIS** [[9]](#и9)

Колекцията QueryString извлича стойностите на променливите в HTTP заявката.

Така

Request.QueryString(name)[index] връща стойността на посочения параметър, с индекс от 1 до броя подадени стойности.

Особеното тук, е че ако не бъде посочен индекс когато параметърът има няколко стойности, то методът ще върне стринг, съдържащ всичките стойности, разделени със запетая.

**Уязвимостта**

С досега наученото можем ясно да види потенциалният проблем. Reverse proxy сървърът е идеално място за филтриране на user input за опасно съдържание. Поради това програмистът може да се осмели да не извършва такава проверка в backend сървърите, с мисълта че проксито ще им подава само сигурни стойности.

Но ако reverse proxy и вътрешните сървъри използват различна технология, има риск проксито да филтрира едни стойности на подадените променливи ,а backend-а да ползва други.

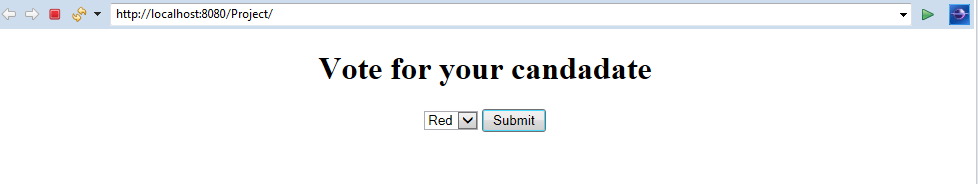
Някои възможни начини HPP да може да заобикаля филтрите:

1. Reverse proxy сървърът проверява само първата стойност на променливата, напр Tomcat, докато задният сървър използва последната, напр PHP.

2. Reverse proxy сървърът филтрира поотделно стойностите на променливата, докато задният сървър ги ползва, след като конкатенира със запетая всичките стойности.

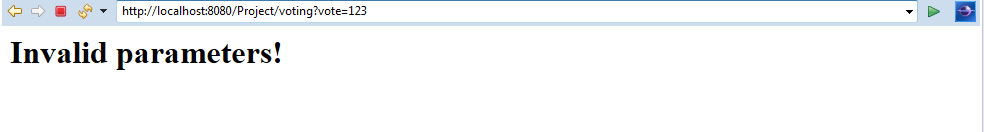
Пример за 1.

Към реферата има примерно уеб приложение за гласуване за един от двама кандидати. В него потребителските заявки с параметър "vote" - за кого се гласува, се подават на Tomcat сървър. В този сървър "MyServlet" играе ролята на филтриращо прокси, и ако стойността на "vote" е допустима, заявката се препраща към PHP сървър - "votebackend.php" , който осъществява връзка с база данни за подадените гласове.

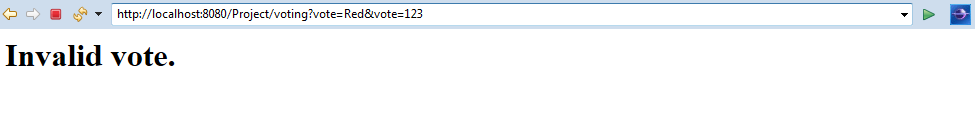
****User < - > MyServlet.java < - > votebackend.php < - > MySQL таблица "results"

****

При опит за подаване на невалидна стойност за "vote", "MyServlet" връща съобщение за грешка.



Проблемът идва, когато за "vote" са подадени няколко стойности, напр vote=Red&vote=123

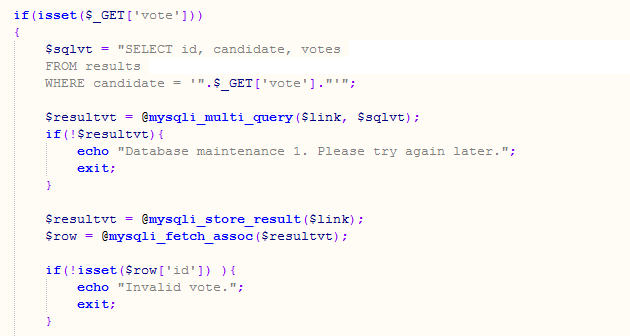


Полученото съобщение идва от "votebackend" , когато подаденият в "vote" кандидат не присъства в базата данни. Филтърът в проксито обаче не би трябвало да допуске такава стойност на параметъра.



"MyServlet" изпраща заявка до от "votebackend" **само** когато параметърът отговаря на посочените условия ( филтриране с whitelist ). Но както беше посочено горе, request.getParameter взима само първата стойност, която в случая е валидна.

Така заявката достига до "votebackend", където $\_GET взема последната стойност, без да я филтрира.



Така приложението става уязвимо на други типове атаки, например SQL Injection:

Get заявката „<http://localhost:8080/Project/voting?vote=Blue&vote=Blue';%20UPDATE%20results%20SET%20votes=0%20WHERE%20candidate='Red>“ превръща заявката до базата данни $sqlvt в

"SELECT id, candidate, votes FROM results WHERE candidate = 'Blue'; UPDATE results SET votes=0 WHERE candidate='Red' ";

Щетите за базата данни могат да са огромни.

**Предпазване от атаката**

От изключително значение е отличното познаване на структурата на уеб приложението.

В зависимост от използваните сървъри е възможно системата да няма тази уязвимост (reverce proxy и backend-а да обработват едни и същи стойности). А както беше разгледано при Perl и ASP, може да има избор как да бъдат приети url параметрите с много стойности.

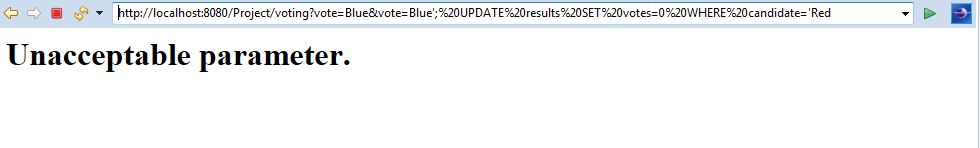
Освен това,тъй като основната функция на проксито е да пренасочи потребителската заявка към някой backend сървър, трябва да се следи за спазването на строг шаблон, при създаването на url-а към вътрешният сървър. Това се явява още една защита, от предаването на нежелани данни назад в системата.

Но най-сигурният начин за предпазване от атаката, е да се прилага филтриране на променливите и във задните сървъри. Много е важно проверката на стойностите да е съобразена с контекста в който те ще бъдат използвани, за да се знае какви типове атаки чрез параметрите могат да засегнат системата.

Например, "votebackend" е податлив на SQL Injection, тъй като тъй като параметър "vote" директно се използва за създаване на заявката. Уязвимостта се премахва, чрез добавяне на whitelist филтър, и за още по сигурно – използване на Prepared Statement.



В резултат, атаката от последният пример бива осуетена.



Както показва този тип атака, ако технологиите нямат стандарт за справяне с определена ситуация, това дава път на уязвимости.

Но когато програмистът разбира чудатостите на езиците с които работи, може да направи приложенията си много по-сигурни.

**Източници**

[1]

<https://en.wikipedia.org/wiki/Reverse_proxy>

<https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/>

<https://avinetworks.com/glossary/reverse-proxy-server/>

<https://www.avast.com/c-what-is-a-reverse-proxy>

[2] OWASP EU09 Poland – HTTP Parameter Pollution – Luca Carettoni, Stefano di Paola

[3] HTTP Parameter Pollution Vulnerabilities in Web Applications – Marco Balduzzi

[4]

<https://www.imperva.com/learn/application-security/http-parameter-pollution/>

<https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/http-parameter-pollution/>

[5] <https://www.php.net/manual/en/reserved.variables.get>

[6] <http://tomcat.apache.org/tomcat-7.0-doc/api/org/apache/catalina/connector/Request.html#getParameterValues(java.lang.String)>

[7] <https://docs.python.org/3/library/urllib.parse.html?highlight=parse_qs#urllib.parse.parse_qs>

[8] <https://metacpan.org/pod/CGI#Fetching-the-names-of-all-the-parameters-passed-to-your-script>

[9] <https://docs.microsoft.com/en-us/previous-versions/iis/6.0-sdk/ms524784(v%3Dvs.90)>

[10] <https://www.acunetix.com/blog/articles/a-fresh-look-on-reverse-proxy-related-attacks/>