Responsible Disclosure

Reflected Cross-Site Scripting

The host regex parameter of the Ganglia Web 3.7.2 application was determined to be vulnerable to Reflected Link Cross Site Scripting (XSS) attacks. These could potentially enable an attacker to trick users into revealing sensitive details such as authentication credentials or access tokens.

XSS is a term used to describe several different methods of attack. Reflected XSS attacks are a means whereby a user can be tricked into executing malicious code by clicking on a link which has been manipulated in such a way as to appear to be that of a trusted site. Such manipulated links are distributed by mass mailings or instant messages and typically would include images taken from the legitimate web site to maximise the trust the recipient might place in them.

Often these links are simply disguised so that a link which reads as, for example, /trusted-server/login.asp in fact points to /malicious-server/fake-web-page.asp. However, in a Reflected XSS attack, these links can be used to execute code such that the address and secure status bars of any browser appear to be valid, making the attack almost indistinguishable from a legitimate site and so increasing the chances of a user being fooled. Of course, the ability to execute code in this manner could allow an attacker to do much more than this. Common attack vectors include stealing session IDs, key logging

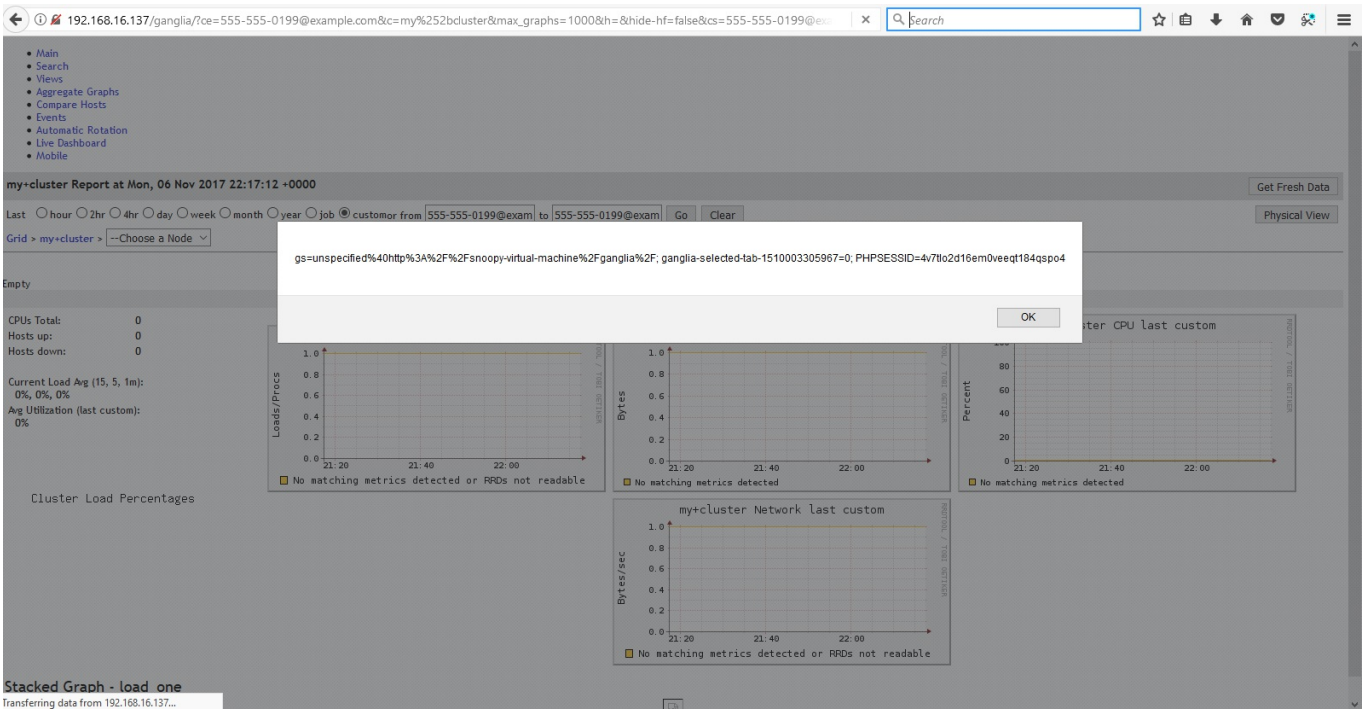
authentication credentials and even effectively taking periodic screen shots of the user’s browsing session.

Many such attacks have been suffered throughout the banking industry, resulting in considerable adverse publicity and loss of public confidence. The parameter vulnerable to this attack can be seen below:

```
GET /ganglia/?ce=555-555-0199@example.com&c=my%252bcluste
r&max_graphs=1000&h=&hide-hf=false&cs=555-555-0199@exampl
e.com&r=2hr&s=descending&tab=m&sh=2&vn=&z=medium&hc=1&hos
t_regex=555-555-0199@example.comk2bse%22%3e%3cscript%3eal
ert(document.cookie)%3c%2fscript%3eiwjf4 HTTP/1.1
Host: 192.168.16.137
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.16.137/ganglia/?ce=555-555-0199@e
xample.com&c=my%2bcluster&max_graphs=1000&h=&hide-hf=fals
e&cs=555-555-0199@example.com&r=2hr&s=descending&tab=m&sh
=2&vn=&z=medium&hc=1&host_regex=555-555-0199@example.com
Cookie: PHPSESSID=4v7tlo2d16em0veeqt184qspo4; gs=unspecif
ied%40
```

In the above HTTP request, the “host_regex” parameter was replaced

with a JavaScript payload `<script>alert(document.cookie)</script>`. This payload will cause the contents of the user's cookie to be displayed in an alert box. This can bee seen in the screenshot below:



Typically, an attacker would send the tampered URL to an unsuspecting user by email. To ensure that this email appeared to be legitimate, the attacker would embed graphics taken from the legitimate web site. This URL could also be Unicode encoded (to disguise embedded code) and embedded into the email. The email would then be spoofed to appear to come from a trusted email address. The user would interact with this object (by clicking on the link), and they would be redirected to the malicious web site.

This type of vulnerability is most commonly exploited in the form of Phishing attacks, typically targeted against banks and otherrespected institutions. Whilst Phishing attacks involve some elements which are

not within control, it is important that all possible steps are taken to minimise vulnerability to such attacks. This is because users may accord web pages a higher degree of trust than other web sites, making it easier for an attacker to execute code on their system.

Solution

It is recommended that the application code be redesigned so that all output to the browser is HTML encoded. This would ensure that even if malicious content was processed by the application, it would always render safely in the user's browser.

To prevent malicious content being accepted by the application, it is also recommended that all input be subjected to strict input validation. This should ensure that any untrusted content is valid and is able to be processed securely by the application. The best way to validate input is to construct a whitelist of valid characters for each input, and check each input for characters which do not comply with the whitelist.

References

Example payload

- http://192.168.16.137/ganglia/?ce=555-555-0199@example.com&c=my%2bcluster&max_graphs=1000&h=&l
  hf=false&cs=555-555-0199@example.com&r=2hr&s=descending&tab=m&sh=2&vn=&
  555-0199@example.comk2bse">

<script>alert(document.cookie)<%2Fscript>iwjf4