Investigation of a Hacked Drupal Site

Exploitation of SA-CORE-2014-005/ CVE-2014-3704 ("Drupaggedon")

Michael Van Dyke, CISSP

11/26/2014

Executive Summary

On October 21, 2014, an attempt to compromise my personal web site was partially successful. The attack was able to delete log entries for October 21, 2014, and was able to add a non-existent user to the administrator role on the web site. The attack apparently failed to actually create the user, however.

The attack exploited an SQL injection vulnerability. This vulnerability was almost certainly the vulnerability describe in Drupal's SA-CORE-2014-005 advisory, released on October 15, 2014. This vulnerability lead to what is referred to in the Drupal community as "Drupaggedon", an event where a massive number of Drupal sites were compromised.

Investigation of the attack involved comparison of the compromised site to a backup of the site made on October 4, 2014. The analysis was performed using standard unix commands, and a Perl script used to break large lines in the mysql backups into manageable chunks. The most important information was located in the "watchdog" table of the site, which contained some of the information that would have been in the missing log file.

Important lessons from this incident

- Prompt patching would have prevented this partial compromise.
- Site administrators should consider changing the administrator role from the default numeric value
- Patching prior to data gathering, although recommended by Drupal.org, significantly complicated the investigation
- Log entries should be sequestered in real time by web sites if they have a high security requirement (remote syslog, etc.)
- The ability of Drupal to access the backend database directly is a fundamental design flaw.

Contents

Background

On October 15, 2014, the Drupal Security Team posted Security Advisory SA-CORE-2014-005 - Drupal core - SQL injection [1] Greg Knaddison[1] of card.com also issued an FAQ on the same day [2]. The FAQ identifies this vulnerability as CVE-2014-3704. Drupal Security Team subsequently released a public service announcement on October 29, 2014 [3].

I operate a Drupal website, with two subordinate sites. These are very low traffic web sites. My wife and I use the main site to post stories we believe might be of interest to our friends and relatives. My wife is a harpist who uses one of the subordinate sites to publicize her harp gigs. I am a volunteer firefighter and user the other subordinate site to post articles about firefighting.

In general, these sites get very few legitimate visitors per day. Actually, they get basically zero legitimate visitors on a typical day.

I followed the instructions in [3] to restore my site on November 2, 2014. At that time, the only indication that the site had been compromised was the loss of the October 21, 2014 log file. I assumed this meant the site had been compromised on that date.

Based on my review, it appears that the main site was partially exploited using the vulnerability described in [1] on October 22, 2014 at 05:20, UST (October 21, 2014 22:20 server time). I say "Partially" exploited because it appears the attacker attempted to create an account on the site, but failed. The attacker attempted to include this account in the administrator role, which succeeded. The exploit thus granted privilege to a non-existent user. This exploit was likely part of a large scale automated attack on multiple sites. There is no evidence that the attacker attempted to log into the account that the attack attempted to create[2].

The database modification was done through SQL injection.

First Indicator of Compromise – Absent Log File

The first indication I had that the server was compromised was the absence of the October 21, 2014 log file. During the data gathering phase of the investigation, I downloaded all log files from the service provider. There simply was no log for October 21, 2014. This means that the log file was never created, or was deleted.

I was not able to determine how the log file was deleted/prevented from being created, however the nature of this attack indicates that a simple "drop table" command would have worked (a classic "Bobby

[1] https://www.drupal.org/u/greggles

[2] Or the attacker did log in, committed acts of evil, and deleted the account and most but not all traces of the activity. Given that the attacker seemed OK with deleting an entire day's log files, it seems unlikely that he/she/it would later exhibit this level of 9partial) finesse.

Tables" problem[3]). Presumably this would have taken effect after the drop tables command had been logged, deleting the log entry along with the rest of the table.

This is substantiated by the fact that the commands to exploit the web server appear very early in the watchdog table (see Third indicator of Compromise – Attack Strings in the watchdog Table).

Second Indicator of Compromise – New Role Assignment

After performing a quick comparison of the October 4, 2014 backup of the site to the compromised site, the inclusion of userID 99992 in the administrator role on the web site leapt out at me. There are whopping two users on, with user Id 1, and (you guessed it) 2.

Third indicator of Compromise – Attack Strings in the watchdog Table

Although the attack deleted the October 21, 2014 log file, entries in the Watchdog table were still in the database. By searching for 99992 in the watchdog table, I was able to find the sql injection string used to attempt to create a user "drplsys" with userID 99992 (see "Discussion" page 5 for more information).

These entries also listed the IP the attack came from, which resolves to a London based hosting provider's network in Kiev, Ukraine. It is unlikely that this is the IP of attacker's machine, but more likely a machine the attacker has compromised or leased anonymously. Less likely, it could be a completely fictitious IP.

I discuss the question of why the attacker did not delete the attack strings and other error messages related to the attack from the watchdog table below, see "Why the attack strings were not deleted", page 6.

Methods

Tools

I investigated this compromise primarily using standard Unix commands (grep, wc, head, tail, less). I wrote one Perl script which simply (arbitrarily) broke long lines into smaller chunks. This was necessary because the individual lines in the mysql database backups could be very long – over 200,000 bytes in most cases. The longest line was over 300,000 bytes. The script initially inserted a line break after every semicolon. There is nothing magic about breaking the input on semicolons, other than that they seemed to show up a lot. I later modified the script to break the file up based on close-parenthesis, which more closely aligned with the file's structure[4].

[3] http://xkcd.com/327/
[4] I started working on a script that fully understood the structure of the Drupal tables, however this would have been more time consuming than performing the analysis by hand.

All analysis was conducted on an Asus G750 laptop running Ubuntu Linux.

Data Analysis

All data analyzed was static. At no time did I fire up the compromised web site, or restore a database from the compromised site.

The primary source of information was the database backup from the primary site.

I got less useful information from comparing the HTML, as most (probably all) changes resulted from the upgrade to Drupal 3.72.

Time Travel

The hosting provider makes historical versions of files available for a limited time. I was able to access the site as it appeared on Nov 1, create a zip file, and down load it. As this feature came to my attention late in the investigation, I have only performed ad hoc analysis on these files. Some ad hoc browsing of the files indicates that the HTML was not impacted. This is unsurprising, as the HTML on a Drupal site is primarily related to the structure of the site, not the content. The “goods” are all in the database, which was the focus of my analysis.

Discussion

This section discusses my conjectures about the attack. It is based on the facts presented above.

Motivation

The attacker almost certainly attacked my site as part of a blitz attack on multiple Drupal installations, with the intent of “pwning” multiple sites for later use to send spam, conduct SEO poisoning, as a launch pad for other attacks or other purposes. It is possible the attacker had no purpose beond the desire to capitalize on the announced vulnerability -- take over as many sites as possible now, decide what to do with them later.

Why my site was attacked relatively late

Reports on line indicate that the attacks against Drupal sites worldwide began within seven hours of the announcement of the vulnerability on October 15, 2014. My site was not attacked until October 21. While six days may not seem long, it is well into the “Drupageddon” timeline. Some reasons for this delay that occur to me are documented below.

My site may indeed have been attacked earlier

It is possible that the site was attacked earlier. I have looked for the indicators I list below (see “Indicators”, page 6) in the logs prior to October 21, 2014 and have not found them. This only means that *this specific* attack probably did not occur. Other attacks may have happened with different indicators.

My site is a low value target

It is possible that the attacker(s) involved in drupageddon were primarily interested in sites in the .com TLD space on the assumption that "that's where the money is". My site is not a .com, and is not involved in any kind of e-commerce. It has no proprietary data. In short, anyone doing reconnaissance of my site would be unlikely to flag it as high value (and it in fact isn't).

Why the attack strings were not deleted

Although the attack strings do not appear in the log *file*, they do appear in the watchdog *table* in the database. I believe this is a result of the timing of packets crossing the Internet from Kiev to Texas (where the hosting provider is located). Most likely, the attacker *issued* three commands in this order:

- Create the new user account (drplsys,99992)
- Add the new user to the administrators group
- "drop tables", or other mass delete command on the watchdog table

Had the commands *executed* in that order on the web site, all traces of the attack strings would be gone. Because of the number of routers between Kiev and Texas, it is possible that the commands arrived out of order. The drop tables sql injection arrived first, causing the watchdog table to be deleted (or cleared). Then the other commands arrived and were logged in the (now mostly empty) watchdog table. Bear in mind, this was almost certainly not someone sitting at a keyboard typing these commands one at a time, but a script that was attacking my site and thousands or millions of others at the same time.

This is supported by the fact that the attack strings appear fairly early in the watchdog table (the first error that appears to be related to the attack is the 34th entry of 1018 total entries). This also accounts for why the command to clear the watchdog table does not appear in the watchdog table – it would be the last entry made before the table was cleared.

Other possible attacks/compromises

This is an analysis of *one* attack on the site. There is no compelling reason to believe that others did not make attempts to attack the site. It is possible that other attacks not only occurred, but perhaps succeeded. It is possible that evidence of those attacks remain in the log files or the database of the compromised site.

Indicators

Web administrators interested in determining if this specific attack has targeted or compromised their site should look for:

- Log entries for failed login attempts with the string "insert into users" (this will match both the attempt to add the user, and the attempt to add the user into the "users_roles" table).

- New users, especially those with very high userIDs. Some web discussions indicate that 99994 has been used in some cases, as well as 99992. I assume the high numbers are to try to avoid collisions with existing users.
- Missing log files.

Lessons

There are a few things to be learned from this, or at least considered.

Prompt Patching

Prompt patching would have prevented this partial compromise. While I have generally gotten by patching the site once per month, this advisory was released as critical and probably warranted more immediate attention. Given the overall low value of this site, the only damage done was the consumption of my time to rebuild the site from backups.

Administrator Role Number

The attack used the default administrator role number of 3. Sites that set their administrator role to a different value would not be susceptible to this attack. Although a sophisticated attacker could do reconnaissance to determine whatever value the administrator role was set to, it is unlikely they would bother.

Gather Data, Then Patch

I followed the advice in the PSA [3] to the letter, on the grounds that they knew more about the compromise than I did. This meant patching immediately, then gathering information. This would have been a win in the (unlikely) case that my site had not been compromised, but was about to be. It hindered my investigation, however, because I spent a lot of time comparing HTML files that had changed – because of the patching.

Late in the investigation I discovered that the hosting provider had a feature in their file manager that allowed me to go back in time (see Time Travel page 5) and retrieve the unpatched version of the files. A quick review of the unpatched site showed that only files related to patching I had done back on October 4 had changed.

Sequester Log Entries

Although through sheer luck some of the log entries for this exploit were still in the database (but not the log file, which apparently never got created), it was clearly the attacker’s intention to delete all evidence. A process to harvest log entries as they are made and sequester them on a remote server would not only (in this case) provide a means to identify how the log file was deleted, it would (more generally) provide a quick and dirty way to find malicious activity, as any difference between the local and remote log files would give away the fact that someone was hiding something.

Drupal Does Not Have Sufficient Separation of the Database From The Web Server

This compromise is the result of inadequate sanitization of user input *combined with* the ability of the web site to directly issue SQL commands that can read and alter the database. This is a fundamental design flaw that makes Drupal inappropriate for any type production site that has a strong requirement for confidentiality, integrity, or availability.

Next Steps

The following are logical next steps to carry the investigation further:

Replicate in a lab setting

Although the information I got from the log files on the compromise seems clear enough, I would like to better understand the suppression/deletion of the log file. I would also have more confidence in my conclusion that the attack was only partially successful if I could understand why the attack string used to try to create the new user did not work.

My conjecture on the timing of the log file deletion also would benefit from some testing. I don’t fully understand why the entries that were in the watchdog table didn’t make it into any log file. In a lab environment, I could better understand the Drupal log generation process.

Assess other possible compromise vectors

A noisy attack such as this one makes me wonder if it isn’t all an attempt to distract from a more subtle compromise.

Identify Reconnaissance Probes

Although it is possible that the attacker acted blindly, sending the attack without knowing if my site ran the vulnerable version of Drupal, or in fact ran Drupal at all, it is more likely that there was some level of reconnaissance of the site prior to the attack. While these probes may be lost with the missing log entries, it is also possible that reconnaissance occurred days before the attack. In that case, evidence may still be in existing log files.

Acknowledgements

I would like to thank the hosting provider support team for discussing the compromise with me via online chat, and especially for showing me how to go back in time to retrieve the pre-patched version of the compromised site.

Information Not Included in this Paper

This report deliberately does not provide the specific attacks strings found, as it is not my purpose to provide explicit directions on how to exploit unpatched web sites.

The attack source IP is of no use as an indicator, as the attacker most likely has compromised multiple sites and can launch the attack from any of them.

The names of my web sites are not relevant to the investigation. Nor is the name of my hosting provider.

Works Cited

[1] "SA-CORE-2014-005 - Drupal core - SQL injection," 15 October 2014. [Online]. Available: https://www.drupal.org/SA-CORE-2014-005. [Accessed 23 November 2014].

[2] G. K. ("greggles"), "FAQ on SA-CORE-2014-005," Card.com, 15 October 2014. [Online]. Available: https://www.drupal.org/drupalsa05FAQ. [Accessed 23 November 2014].

[3] Drupal Security Team, "Drupal Core - Highly Critical - Public Service announcement - PSA-2014-003," 29 October 2014. [Online]. Available: https://www.drupal.org/PSA-2014-003. [Accessed 23 November 2014].