Join the Discussion
Connect

Web Security Tools²: skipfish and iScanner

By Russ McRee – ISSA member, Puget Sound (Seattle), USA Chapter

Prerequisites

Linux GCC compiler or Cygwin GCC environment to compile skipfish

Ruby to run iScanner

Website and web application security concerns are many, the threats are endless, and the more tools you have in your arsenal the more capable your defense may be.

Two attack vectors of regular focus are as follows:

1. Attacks against web application security flaws. The OWASP Top 10 is an excellent indicator of how widespread and prevalent these issues are, including script injection, server configuration issues, as well as cross-site scripting and request forgery.
2. Server configuration errors (weak or poorly implemented permissions settings) can a lead to attacks where malicious code is embedded within website code, often redirecting victims to additional sites propagating malware.

We'll focus on Michal Zalewski's *skipfish* to focus on #1, and *iScanner* from isecur1ty.org to address #2.

Skipfish is an "active web application security reconnaissance tool that prepares an interactive sitemap for the targeted site by carrying out a recursive crawl and dictionary-based probes." The resulting map is then populated with the findings of a number of active security checks resulting in a final report meant to serve as a foundation for professional web application security assessments, a target list if you will.[1]

Michal indicates that skipfish is a bit different from what people may be used to, for example, it tries to exhaustively brute-force all directories unless instructed otherwise (while most other scanners "automagically" bail out with no way for you to actually understand the resulting coverage). This sometimes confuses some users who point skipfish at a site with, as an example, 3,000 fuzzable directories, and expect it to complete the scan soon.

Simply, skipfish is useful for comprehensive reconnaissance and should not be expected to gather you the quick, down and dirty results you may achieve with other scanners. Refer to "Known Issues" for further details.[2]

iScanner is a Ruby-based tool that "detects and removes malicious code and webpages malware from your website with automated ease. iScanner will not only show you the infected files from your server but it's also able to clean these files by removing the malware code from the infected files."[3] iScanner includes extensive logging, support for email reports, and backup of analyzed files, as well as speed and flexibility.

Installing and configuring skipfish

Download the skipfish[4] package, unpack it, and `cd` accordingly.

Installation may include various dependencies specific to the system on which you choose to install skipfish.

As I have moved away from Ubuntu back to Debian for stability's sake, I needed to install the `libcurl4-openssl-dev` package to meet the make requirements for OpenSSL (including development headers).

You may need build-essential to fulfill `gcc` requirements, as well as `zlib` and `libidn`.

See Problem #1 of Known Issues should you require more detail.[5]

Assuming all dependencies are met, execute make, and the skipfish binary should be evident in the installation directory.

Copy a dictionary file of your choosing from the dictionaries directory to the skipfish root directory and rename it *skipfish.wl*.

- **Extension-only dictionary** (extensions-only.wl): 90 common file extensions, and no other keywords; must be used in conjunction with `-Y`
- **Basic extensions dictionary** (minimal.wl): 25 file extensions and 1,700 +/- hand-picked keywords
- **Standard extensions dictionary** (default.wl): 60 file extensions and 1,700 +/- hand-picked keywords

1 http://code.google.com/p/skipfish/wiki/SkipfishDoc.

2 http://code.google.com/p/skipfish/wiki/KnownIssues.

3 http://iscanner.isecur1ty.org.

4 http://code.google.com/p/skipfish/downloads/list.

5 http://code.google.com/p/skipfish/wiki/KnownIssues.

- **Complete extensions dictionary** (complete.wl): 90 file extensions and 1,700 +/- hand-picked keywords

You can also opt for an empty dictionary file in conjunction with `–W`; using it with `–L` inhibits all brute-force testing and results in an orderly, link-based crawl.

`./skipfish -h` will define all options for you.

Installing and configuring iScanner

The primary dependency for iScanner is Ruby. If you don't have Ruby, execute `sudo apt-get install ruby` or `yum install ruby`, depending on your *nix system preferences.

Thereafter, a simple `sudo ./installer –i` from the directory in which you unpacked iScanner and you're ready to roll. I don't care for the default installation location of `/etc/iscanner`; something about running apps out of `/etc` bugs me. I changed the installation directory as defined in the installer ruby script to my tools directory. iScanner documentation indicates that you should be able to run `sudo ./installer -i -d /home/rmcree/tools/iscanner` to accomplish the same goal, but it resulted in Ruby errors for me.

Using skipfish

I run skipfish against web applications for which I conduct vulnerability assessments against installations in my private test environment. As the skipfish project documentation says, don't be evil.[6]

I'll be honest with you, skipfish is not sexy, nor is meant to be. It's a performer: code light, easy to use, well-proven security checks, and clean reporting. It is not fast, in the sense that your results come quickly. You must be patient, young Jedi.

With the complete dictionary enabled, I've had scans against local installations still exceed 24 hours.

That said the resulting "interactive sitemap" is an extremely useful part of initial reconnaissance.

For an application that requires authentication, and whose results I want stored in *testapp*, I executed the following (See Figure 1):

```
./skipfish –A admin:admin –o testapp http://
localhost/testapp
```

You can further define options for authentication and access, crawl scope, reporting, and dictionary management. Again, use `./skipfish –h` for further detail.

Skipfish won't give you exact attack strings in the results; it will give you a comprehensive list of opportunities for further review. Simply browse *index.html* in your results directory (See Figure 2).

Skipfish is well-documented and Michal is quite candid about known limitations and features wishlist. Missing from the current code base are buffer overflow checks, specific PHP tests, scan resumption, search engine integration, and others. That should not stop you from making use of this excellent tool. As with all projects of this nature, if something is missing that you'd like to see incorporated, feel free to develop and contribute!

6 http://code.google.com/p/skipfish/wiki/SkipfishDoc.

```
skipfish version 1.34b by <lcamtuf@google.com>

  - localhost -

Scan statistics
---------------

      Scan time : 1:10:37.0921
  HTTP requests : 359431 sent (84.93/s), 327777.12 kB in, 104348.65 kB out (101.97 kB/s)
    Compression : 188356.42 kB in, 396142.09 kB out (35.55% gain)
 HTTP exceptions : 0 net errors, 0 proto errors, 0 retried, 0 drops
 TCP connections : 3565 total (101.85 req/conn)
  TCP exceptions : 0 failures, 0 timeouts, 1 purged
  External links : 1309 skipped
    Reqs pending : 3680

Database statistics
-------------------

         Pivots : 853 total, 379 done (44.43%)
    In progress : 370 pending, 58 init, 26 attacks, 20 dict
  Missing nodes : 47 spotted
     Node types : 1 serv, 93 dir, 52 file, 4 pinfo, 355 unkn, 105 par, 243 val
   Issues found : 39 info, 3 warn, 183 low, 162 medium, 1 high impact
      Dict size : 2271 words (272 new), 39 extensions, 256 candidates

[!] Scan aborted by user, bailing out!
[+] Wordlist 'skipfish.wl' updated (272 new words added).
[+] Copying static resources...
[+] Sorting and annotating crawl nodes: 853
[+] Looking for duplicate entries: 853
[+] Counting unique issues: 543
[+] Writing scan description...
[+] Counting unique issues: 853
[+] Generating summary views...
[+] Report saved to '     /index.html' [0x63e20174].
[+] This was a great day for science!
```

Figure 1 – skipfish run

Figure 2 – skipfish findings for further assessment

Using iScanner

iScanner does not currently scan remote sites; you must have the site code you wish to review stored locally. Use the likes of wget to pull down HTML or if you're scanning code for a site you manage, use sftp to pull down a local copy to your analysis system. There's always the option to run iScanner on the actual web server if Ruby is available, but I don't really care for this option as iScanner will actively remove code, depending on how you run it.

I grabbed a real-world example of malicious JavaScript from a recent investigation and buried it in a test website on one of my local systems.

Run iScanner as follows for a first pass, modifying for code directory and log output to suit your preferences:

```
sudo ./iscanner –f /var/www/testsite –o
toolsmith.log
```

Review the results of the log file and ensure no false positive results. My tests found that iScanner will identify certain JavaScript as "code from remote source detected." Such code may be perfectly acceptable, as was the case in this test. Therefore, always check your results before letting iScanner cleanup for you. I strongly recommend against running iScanner with the auto-clean option set (`-a`). Instead, after your first pass as described above, narrow down your cleaning targets. Figure 3 shows the results of the first run.

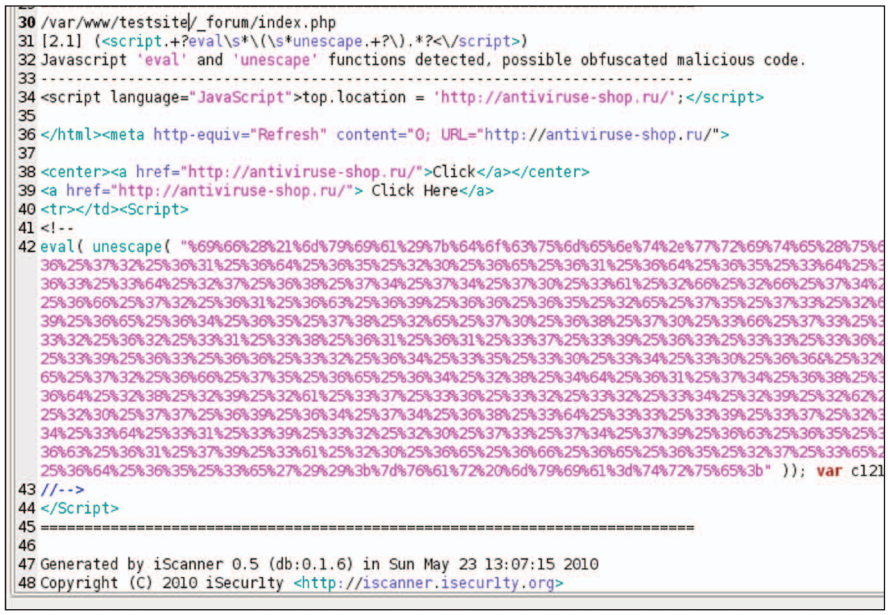

Figure 3 – iScanner results for further review

The obfuscated malicious code was discovered in `/var/www/testsite/_forum/index.php`.

As you've confirmed the code as malicious and wish to remove it, execute `sudo ./iscanner –F /var/www/testsite/_forum/index.php –o toolsmithClean.log`.

This will confirm your initial test run and reduce output to malicious findings only.

Then execute `sudo ./iscanner –c toolsmithClean.log` and the obfuscated JavaScript will be removed from `/var/www/testsite/_forum/index.php`. You always have the option to take this step manually, but the fact that cleanup is built into iScanner is a nice feature.

Like it's *toolsmith* partner skipfish, you can dump all iScanner options with `./iscanner –h`.[7]

Additional options such as version and database update, backup and restore, and extension-specific scans are available.

iScanner makes use of a signatures database; a `.db` file that consists of regular expressions crafted to detect malicious code, potentially obfuscated and/or encoded. The string that identified my intentionally hidden sample was (`<script.+?eval\s*\(\s*unescape.+?\).*?<\/script>`), as clearly indicated in Figure 3.

You can enhance the signature database with detective regex of your own, and check for updates from the developer via `–u` (version updates via `–U`).

In conclusion

Skipfish and iScanner, albeit quite different, are both definite additions for your toolkits.

Reduction of web application security flaws as well as the identification and removal of obfuscated malcode are important ongoing processes as part of your proactive and reactive defensive measures.

I recommend both these tools and appreciate the customization options via the dictionary files in skipfish and the signatures database in iScanner.

Make good use of them both.

Cheers…until next month.

Acknowledgements

—Michal Zalewski, for skipfish details

—Bryan Casper, for additional iScanner testing

About the Author

Russ McRee, GCIH, GCFA, GPEN, CISSP, is team leader and senior security analyst for Microsoft's Online Services Security Incident Management team. As an advocate of a holistic approach to information security, Russ' website is holisticinfosec.org. Contact him at russ@holisticinfosec.org.

7 http://iscanner.isecur1ty.org/documentation.html.