The Ratproxy security scanner looks for vulnerabilities in web applications

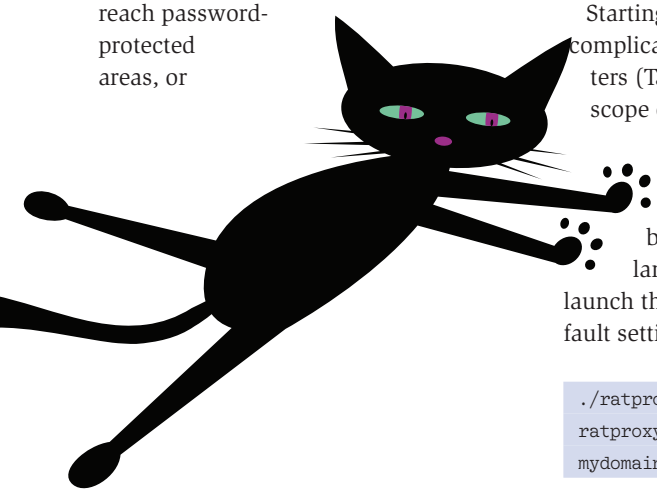
RAT CATCHER

Google's Ratproxy is a free testing tool that searches for security problems in web applications. **BY TIM SCHÜRMANN**

Several test suites help you look for vulnerabilities in web-based applications [1] [2], but many of these applications are expensive or difficult to use. Wouldn't it be nice just to press a button to find out what vulnerabilities exist in your own software – along with a line reference to help you find the problems in the source code?

Ratproxy [1] is a tiny but powerful tool with a simple approach to searching for problems in web applications. The Ratproxy security testing tool originated in the development labs Google, where it was created to test Google's own applications. In July 2008, the company decided to release the current version to the general public under the Apache License 2.0.

Google describes Ratproxy as a "semi-automatic, largely passive web application security audit tool." Lurking behind this cryptic description is a tool with a simple purpose: Ratproxy sniffs communications between the browser and the application, logging the data stream and checking the log for known issues, risks, and vulnerabilities. Developers can launch Ratproxy and watch the output. Ratproxy reaches places that competitive products find difficult to access. For example, other tools might find it hard to reach password-protected areas, or they could stumble over some forwarding scenarios. In the same way, Ratproxy cleverly works around Javascript issues. Legacy test tools employ guess work to discover which function will be used next. This problem occurs particularly in GUI testing, wherein users typically have a choice of various buttons and menus. Ratproxy, on the other hand, simply waits to see what the user does next in the browser.



Because Ratproxy does not cause a noticeable increase in network traffic, it even lets you check applications that are deployed in production environments. (Other scanners launch DOS or cross-site scripting attacks that are likely to bring a production system to its knees.)

Setting the Mousetrap

Deploying Ratproxy is simple: Just download the source code from the homepage and run *make* to build the dozen or so source files. The tool does not require a *configure* script or have any major dependencies. What you do need are the libcrypto and libssl libraries (typically supplied as part of the OpenSSL distribution) and corresponding headers.

Starting the test tool is slightly more complicated: No fewer than 22 parameters (Table 1) govern the nature and scope of the tests Ratproxy performs. The parameters are also responsible for defining the level of detail to output. To avoid being plowed under in an avalanche of messages when you first launch the program, start with the default settings:

```
./ratproxy -v /tmp -w ➥
ratproxy.log -d ➥
mydomain.com -lfscm
```

This command points Ratproxy at the web application in the *mydomain.com* domain. Ratproxy will ignore any URLs not on this server. (This approach is a way of making sure that Ratproxy will not run off and accidentally test external ad sites.) The http traffic sniffed by Ratproxy is dumped into a multitude of tiny files in the temporary directory (*-v /tmp*), whereas the analysis of the results – that is, the information you are actually interested in – is stored in *ratproxy.log*. The *Ratproxy Parameters* box explains the Pandora's box of command-line options.

If you prefer a full broadside, you can change the parameters as follows:

```
./ratproxy -v /tmp ➥
-w ratproxy.log -d ➥
mydomain.com -lextifscgjm
```

Shadowing the User

Ratproxy's interactive orientation has several benefits, but it is also the tool's major deficiency. If the user does not execute a function, Ratproxy does not test it. Before you launch Ratproxy, you should think carefully about which parts of the web application you want to test – and in which order.

Illustrations on this page: Anna Telnova, Fotolia

Figure 1: To point Firefox to Ratproxy, first select Edit | Settings in the main menu; select Advanced, choose the Network tab, and click on the Settings button. In the Connection Settings dialog box, enable Manual proxy configuration, and type the settings for the proxy configuration.

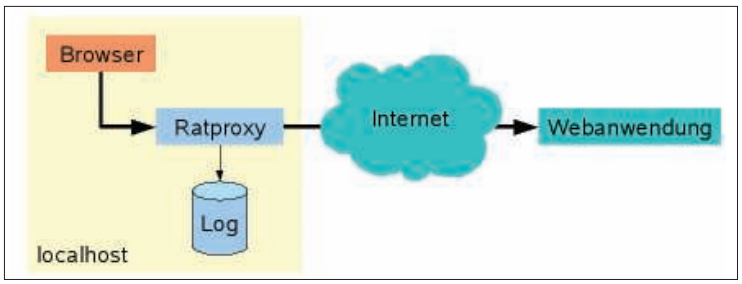


Figure 2: Ratproxy sniffing traffic.

The optional duo *-XC* (note the uppercase letter in the command name), releases Ratproxy from its passive role. Once released, Ratproxy will check to see how well your web application withstands XSS and XSRF attacks (*-X*), and it will repeat requests with modified parameters (*-C*).

If the web application returns Flash objects, Ratproxy can disassemble and analyze them. Ratproxy relies on the Flare ActionScript decompiler for this; unfortunately, Flare is only available as a prebuilt closed source application. By default, Ratproxy supports execution on x86 processors. A version for 64-bit Linux is available on the Flare homepage [4]. First you must download the file, unpack it, and store the results in *flare-dist*.

Connected

Once you see the message *Accepting connections on port 8080/tcp (local only)*, you know that the test tool is listening on port 8080 for incoming browser requests. The next step is to set up the browser to direct all communications via Ratproxy. The easiest way of doing this is to enter this port as a proxy on your own machine (*127.0.0.1* or *localhost*) (Figure 1).

This tells the browser to forward all requests to *localhost:8080*, where Ratproxy will analyze the requests before passing them on to the web application (Figure 2). Because the test tool sniffs traffic passively, all of this is absolutely transparent and has only a minimal effect on execution speed. The *-X* and *-C* parameters, however, are an exception to this rule. They tell Ratproxy to switch to "disruptive mode" and actively interfere with communications. (The effects of these parameters will vary.)

If you use a genuine proxy to access the web, which is the case in many corporate environments, you need to pass the *-P host:port* parameter to Ratproxy, in which *host* and *port* represent the data for your proxy. This feature means you can deploy Ratproxy as part of a chain of other test tools.

Throughput

The next step is to access the web application in your browser and work in the normal way. To avoid interference from other sources, Google recommends closing all other browser windows and flushing the browser cache before you start. Ratproxy will now monitor all your actions and log them in *ratproxy.log*.

In the case of SSL-encrypted data, Ratproxy will replace the certificate served up by the web application with its own. A good browser will alert you to this. To carry on with the test, you must accept the new certificate. The Ratproxy documentation [1] warns against storing the certificate permanently in your browser. After all, everyone who downloads Ratproxy knows the certificate. Because Rat-

Table 1: Ratproxy Parameters

Parameter	Meaning
-l	By default, Ratproxy uses checksums to compare websites. The *-l* parameter enables a less strict method.
-f	Ratproxy will also inspect Flash applications; if you set the *-v* parameter, it disassembles them for a more detailed analysis.
-s	All POST requests are dumped to the logfile.
-c	Remembers pages that set cookies, independently of whether this represents a security risk.
-m	Ratproxy records all content outside of the test domain. Without this parameter, only remotely linked scripts and style sheets are logged. This assumes that you set the *-d* parameter.
-e	Inspects caching more closely.
-x	Logs all URLs that could be useful for further (manual) XSS tests.
-t	By default, Ratproxy logs any directory traversal vulnerabilities . This parameter allows less likely candidates that you could use for manual analysis.
-i	Logs all PNG files returned; PNG files have been misused for XSS attacks in the past (older versions of Internet Explorer are prone to this in particular).
-g	Extends the XSRF tests to include GET requests.
-j	Enables detection of risky JavaScript constructions such as *eval()* calls.
-X	Ratproxy switches to active mode and tests web applications for vulnerability to XSS and XSRF attacks.
-C	Ratproxy repeats some requests with modified parameters.

A complete and comprehensive list of parameters is itemized in the documentation [1].

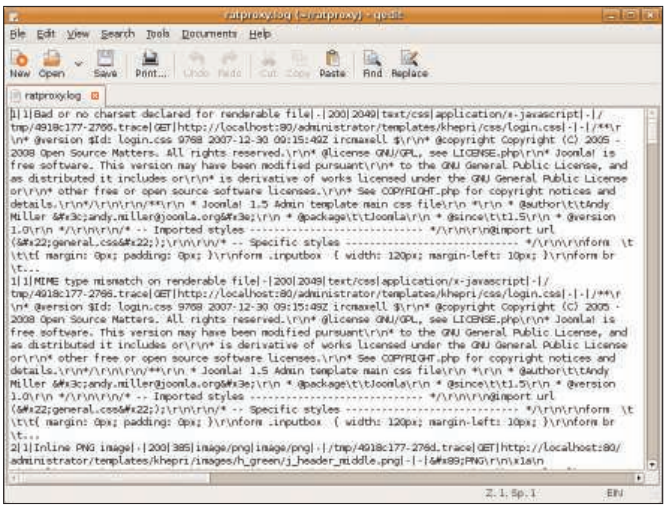


Figure 3: Ratproxy's logfile is not exactly intuitive.

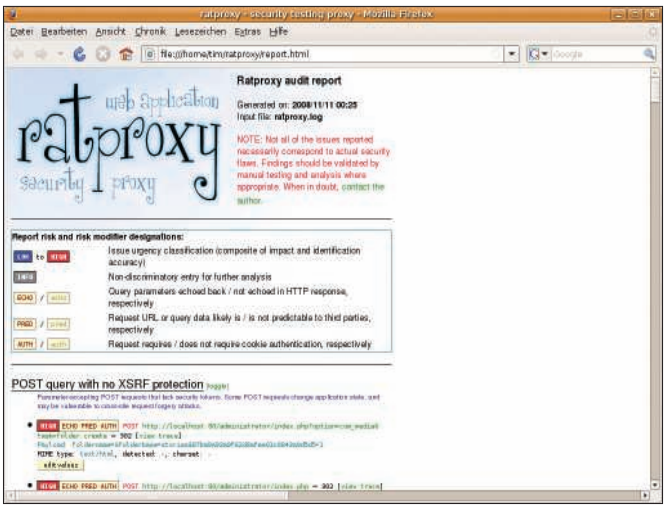


Figure 4: Ratproxy provides a report with results of the testing.

proxy forces a certificate on you, another problem appears: Ratproxy negotiates all further steps with the web application, so you can't be 100 percent certain that you are talking to your own server. Thus, you should avoid using critical (administrative) accounts or entering sensitive data while being monitored by the tool.

On top of this, you should resist the temptation to use *wget* to feed the website to Ratproxy. Most of Ratproxy's tests rely on user interaction and would simply be dropped in the case or a *wget* command.

What the Rat Catcher Reveals

Ratproxy checks the dialog for the following:

- standards compliance, such as the correct use of MIME types (e.g., has a GIF image been served up as *image/jpeg*?)
- insecure responses, particularly with JSON and similar data formats
- cross-site scripting (XSS) attack vectors
- cross-site request forgery (XSRF) attack vectors; Ratproxy focuses in particular on embedded security tokens and predictable URLs
- data injection vectors, such as SQL injection
- risky JavaScript, OGNL and Java constructions
- incorrect use of cookies
- suspicious Flash objects
- directory traversal vectors
- incorrect use of caching
- suspicious redirects

The *messages.list* file supplied with the source code archive gives you details of the problems Ratproxy logs.

The Showdown

Pressing Ctrl + C terminates the Ratproxy test. The results of the analysis land in the slightly cryptic *ratproxy.log* file, which is designed for easy machine readability and for cooperation with *grep* (Figure 3). Until new tools appear, you can use the *ratproxy-report.sh* script to generate a more intuitive HTML report:

```
./ratproxy-report.sh ➥
ratproxy.log > ➥
report.html
```

The report looks like that in Figure 4: The list presents the problems identified by Ratproxy, sorted by type and importance. Critical security risks are highlighted with a neon red *HIGH*. *Toggle* shows or hides the messages in a specific section, and *view trace* opens the trace (i.e., the sniffed communications) from the *tmp* directory.

At this point, the user is left to interpret the results. To do so, you need expert knowledge of both computer security and forensics and details of the application you are testing. After all, it makes little sense for Ratproxy to warn you about a potential cross-site scripting risk if you are unable to close the gap. In other cases, Ratproxy lists generic issues that do not necessarily represent a security risk.

Conclusions

Because Ratproxy works entirely autonomously, you cannot inject your own test data into the web application to confirm your suspicions. Ratproxy can only report on the vulnerabilities it detects in the parts of the web application it actually investigates. (See the box titled "What the Rat Catcher Reveals.") The developers are aware that their product is not perfect, and they ask for suggestions, improvements, and details of any security issues Ratproxy fails to identify.

Remember that Ratproxy is still beta. Don't be surprised to see some false positives, and don't rely on Ratproxy to the exclusion of all other tools. If you are willing to work around the quirks, Ratproxy it is still a useful addition to your security testing toolbox.

Ratproxy is still far from being a panacea. It does not give you a full list of unresolved vulnerabilities, nor does it help you resolve the issues it detects. Interpreting the results requires expert knowledge of web security.

What Ratproxy does do is reliably point you in the direction of potential issues, vulnerabilities, and poor code. If Google continues to refine its tool and can attract third-party vendors to dock at Ratproxy's open interfaces, Ratproxy could develop into a test jewel for web applications. ■

INFO

[1] Ratproxy: *http://code.google.com/p/ratproxy*

[2] Chorizo: *https://chorizo-scanner.com*

[3] Burp Suite: *http://portswigger.net/suite*

[4] Flash decompiler Flare: *http://www.nowrap.de/flare.html*