RUST: The Reusable Security Toolkit*

Chaitanya Atreya*
Adobe Systems
atreya@adobe.com

Adam Aviv†
University of Pennsylvania
aviv@cis.upenn.edu

Maritza Johnson
Columbia University
maritzaj@cs.columbia.edu

Mariana Raykova
Columbia University
mariana@cs.columbia.edu

Steven M. Bellovin
Columbia University
smb@cs.columbia.edu

Gail Kaiser
Columbia University
kaiser@cs.columbia.edu

ABSTRACT

We describe the design of a reusable toolkit for testing anti-phishing technologies. It is based on web bugs and a set of small, simple tools. We show that its existence would have simplified the design of other studies in the field.

1. INTRODUCTION

Testing an anti-phishing technology is time-consuming. One must build real and spoofed websites that employ the technology, carefully devise a test scenario, recruit subjects, run sessions, administer questionnaires, evaluate behavior, analyze collected data, etc. Certain aspects of this process are irreducible. To test a technology, for example, it has to be running *somewhere*. Other aspects, though, are more amenable to automation. In particular, a user's actual click-through behavior can be captured by various mechanisms including modifying the browser or mailer, modifying the web server or scripts, or log file analysis. Still, these can be difficult to prepare as well, if any new anti-phishing technology requires custom software to do the monitoring or analysis.

Instead, we have built a toolkit to simplify the task of preparing a testing environment. Very few changes are needed to the web sites under test. More or less any web site and/or authentication technology can be evaluated for security with little effort. In addition the toolkit provides a test environment that is more reliably uniform. This is an important scientific benefit when gathering empirical evidence: the results of two different studies can be compared much more easily when the testing conditions are the same.

We have used our toolkit for small-scale studies [3] of Microsoft's CardSpace [6] and Verisign's Secure Letterhead [2]. The prototype code we developed considerably simplifies instrumenting web sites and running tests.

*This work supported by a grant by the FSTC.
*Work done while at Columbia University
†Work done while at Columbia University

Symposium on Usable Privacy and Security (SOUPS) 2008, July 23–25, 2008, Pittsburgh, PA USA

2. TOOLKIT STRUCTURE

There are several components to our toolkit. ones are

TinyHTTPProxy Acts as an HTTP proxy and URL rewriter

ProxyBridge Terminate SSL/TLS connections

Mailer.sh Mail generator

Web Bugs Instrument web pages

Logger Generates standardized log files

2.1 TinyHTTPProxy

Test subjects use standard desktop computers with a browser that supports Javascript and HTTP proxies. The browsers are configured to accept Javascript and to send all web page requests to our proxy.The also have mailers. TinyHTTPProxy (described below) receives user web page request. It rewrites them and forwards them to ProxyBridge. ProxyBridge, in turn, logs significant details and passes the requests to the actual web server.

TinyHTTPProxy terminates connections from the browser.[1] In our setup, it provides us with our own Internet namespace and it lets us rewrite portions of the URL to send requests for different sites to different ports. The namespace issue is the most crucial for us. We want sample URLs to say things like `www.citibank.com` or `www.evilhackerdudez.org`, without requiring the involvement of the real web sites. TinyHTTPProxy uses a simple configuration file to describe these rewrites.

A specialized protocol is used for communications between TinyHTTPProxy and ProxyBridge. The primary purpose of this protocol is to communicate the client's IP address; we use that to distinguish between different test subjects.

2.2 ProxyBridge

ProxyBridge serves three main purposes: it terminates SSL/TLS connections, it rewrites the directory name portion of URLs, and it generates log messages for references to web bugs.

In normal Web environments, SSL or TLS connections from browsers are handled by web servers. This is problematic when trying to serve multiple web sites from a single server. We need to terminate the SSL or TLS connection before we pass the request to the web server. We accomplish the separation by having ProxyBridges running on multiple

[1]See `http://www.okisoft.co.jp/esc/python/proxy/`.

port numbers; the redirection by TinyHTTPProxy accomplishes that nicely.

The second function of ProxyBridge is rewriting the pathname portion of URLs. A proper simulation of a financial institution's web site demands that the proper pathnames appear in the URL bar; however, these may not match the pathnames in the simulated site.

ProxyBridge also generates log file entries for web bugs and needs to know the subject's web browser's IP address to permit proper identification of each test. Only TinyHTTPProxy knows this so ProxyBridge forks and creates a new instance of itself for each client IP address, with a separate port number.

2.3 Mail Generator

Our mail generator is rather simple, it builds standard MIME-formatted messages. In addition, two other simple scripts are used. One script sends emails at predetermined time intervals and the remaining script speaks SMTP to submit the mail.

2.4 Web Bugs

When collecting data on how test subjects respond to phishing emails we are interested in web pages related to the login process, knowing the order of the web pages visited reveals whether or not the user fell for the attempted phishing attack. Additionally, we are also interested in how long they are on a specific page, this information can be used when analyzing data for the stages of interaction that need to be reworked in future iterations. We thus instrument pages of interest by inserting a "web bug". A web bug is reference to a small image, typically a 1 pixel by 1 pixel transparent GIF image. We use Javascript to generate dynamic references to the web bug. The "test" field indicates which task the web page is associated with in the session, in cases where the same site is used in evaluating a technology in the same session.

2.5 Logging

The logger is a multi-threaded server that receives annotated URL requests from the ProxyBridge. It correlates requests for particular web bugs to produce simple page dwell times for each subject. That is, it converts a sequence of requests for a given page's bug to a simple record stating how long the subject was on a particular page.

Each request generated by the web bug on a web page produces an entry. The first line is the GET request for the gif web bug. Included in the request is: a numeric value to indicate the amount of time spent on the page so far in the time variable; the IP address string which can be used after the study to identify concurrent test subjects based on which machine they completed their tasks on; a value named test that represents which task the web bug is associated with, this will be useful when the same web page is used for a number of tasks.

3. OTHER APPROACHES

Many usability evaluations have been conducted that require simulating aspects of phishing attacks in a controlled lab environment. In most cases components of RUST would have been useful had they been available at the time. In *Gathering Evidence* [5], users were presented with a set of websites and asked to label each as real or fake. The primary goal was to determine what information users employed to determine a web site's identity. The relevant components of RUST in this case would have been the web proxy service and the logger.

Dhamija et al. [1] conducted a similar study that presented participants with a set of web pages and asked which were fake. The web proxy component of RUST could have been used for serving the spoofed pages and the real pages could have been accessed normally by not including a translation to the configuration file for that domain. Also, it would have been interesting if the study had collected data on how long each user spent on a web page. Because users did not enter credentials on any web site, it wouldn't be relevant to collect that data, but it would be interesting to know how long it took a participant to reach their decision.

In 2006 SOUPS made toolkits available online for each of the papers in the proceedings that presented results from a user study [4]. The available material is not reusable components but rather more detailed descriptions of what they implemented and briefly discuss in the papers.

4. CONCLUSIONS

We have described the design and design philosophy of our security toolkit. We assert that the toolkit is fairly simple to configure and our design has bought us a great deal of flexibility. We can support any web browser and any web server. We can simulate any site, using any standard browser and any web server. All of the necessary components can run on a single machine, without any requirement for special network configuration. Rather than build large, complex, monolithic applications, we built a set of small, simple tools that were well-adapted to their purpose.

5. REFERENCES

[1] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590, New York, NY, USA, 2006. ACM Press.

[2] P. Hallam-Baker. Secure internet letterhead. Security for human ends. In *W3C Workshop on Transparency and Usability of Web Authentication*, 2006.

[3] M. L. Johnson, C. Atreya, A. Aviv, M. Raykova, S. M. Bellovin, and G. E. Kaiser. Modifying evaluation frameworks for user studies with deceit and attack, 2008. In submission.

[4] SOUPS. Security user studies workshop: User study toolkits. `http://cups.cs.cmu.edu/soups/2006/workshop-kits/kits.html`, 2006.

[5] T. Whalen and K. M. Inkpen. Gathering evidence: Use of visual security cues in web browsers. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 137–144, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[6] Windows. Cardspace. `http://msdn2.microsoft.com/en-us/netframework/aa663320.aspx`.