Edith Cowan University
Research Online

Australian Information Security Management Conference

Security Research Institute Conferences

2010

Development and Evaluation of a Secure Web Gateway Using Existing ICAP Open Source Tools

Michael Pearce
University of Canterbury

Ray Hunt
University of Canterbury

Originally published in the Proceedings of the 8th Australian Information Security Mangement Conference, Edith Cowan University, Perth Western Australia, 30th November 2010
This Conference Proceeding is posted at Research Online.
http://ro.ecu.edu.au/ism/96

Development and Evaluation of a Secure Web Gateway Using Existing ICAP Open Source Tools

Michael Pearce and Ray Hunt
Computer Security and Forensics Group
Department of Computer Science and Software Engineering,
University of Canterbury, Christchurch, New Zealand
michael.pearce@canterbury.ac.nz
ray.hunt@canterbury.ac.nz

Abstract

This work in progress paper discusses the development and evaluation of an open source secure web gateway. The proof of concept system uses a combination of open source software (including the Greasyspoon ICAP Server, Squid HTTP proxy, and Clam Antivirus) to perform the various security tasks that range from simple (such as passive content insertion) to more advanced (such as active content alteration) by modules installed on the server. After discussing the makeup of the proof of concept system we discuss our evaluation methodology for both effectiveness and performance. The effectiveness was tested using comparative analysis of groups of self-browsing high interaction client honeypots (employing a variety of security measures) and recording different system alteration rates. Performance was tested across a wide range of variables to determine the failure conditions and optimal set up for the components used.

Keywords

Secure web gateway, ICAP server, honeypots, Squid HTTP proxy, Clam Antivirus, security architecture

INTRODUCTION

There has been a trend in recent years of decreasing use of dedicated desktop applications in favour of increased use of web based applications and browser based tools for both home and enterprise use. For both classes of users the security of these web applications has become more important for their day to day affairs, and even their critical business. The secure undertaking of affairs online is dependent upon many different factors, but the web browser is the common link, and it is especially important because it is the single most used class of software on computers [Beauvisage, 2009].

Due to the way web browsers are designed to work (in an untrusted environment running untrusted code and unchecked data) they can be exposed to code from a very large number of sources every day. For instance if a user goes to only 20 web sites in a day, each containing code for advertising, widgets and included media, then the user could be exposed to code from hundreds to thousands of subdomains. Consider, at the time of writing this the main page on cnn.com had around 60 JavaScript elements (including inline) from 12 different subdomains, of which only 3 appeared to be CNN controlled. The others included social networking sites, advertising sites, survey delivery sites, and content delivery partner sites. If only one of those sources is compromised then the user is at risk. Recent examples of this occurring include websites from The New York Times [NYT, 2009], Fox News [Danchev, 2010], Whitepages [McAffe, 2010], and Gawker [Poulsen, 2009] websites. Thus, while prevalence of infection in legitimate sites is seemingly low [Keats, 2009], the sheer number of code sources makes exposure to malicious code far more likely.

There are two main risks from compromise of untrusted code, it can mislead users, or it can exploit their software, and both are damaging. At present there are three main approaches to protecting vulnerable web users from these risks. These shall be discussed in the next section

BACKGROUND AND RELATED RESEARCH/SOLUTIONS

The methods of mitigating web-based threats are threefold: user notification, content blocking, and content sanitisation. The delivery methods for these are quite different however between home and enterprise environments. The most prevalent home user solutions are based upon the end user computer system - Improved web browsers, security focused browser add-ons, and antivirus and anti malware software. However, many enterprise solutions do (solely) not reside on the end user system; they use centrally managed devices (so-called *secure web gateways*) placed at network entry points to monitor all incoming and outgoing content.

Both of these approaches have weaknesses making them imperfect for most home and small business users however. The client based solutions reside upon the system itself and often require the end user to understand how to keep the system

functioning correctly, whilst the server based system are often expensive to deploy and maintain and often leave little control to the end user should they wish to not perform the default operations.

It would be very useful if a system existed that allowed individuals and small organisations to gain many of the benefits of a secure web gateway without requiring a large capital and operating cost as the enterprise systems require. Furthermore, often they do not need the large degree of central management that these systems offer, as a higher degree of trust is usual in small organisations.
A basic secure web gateway is an application layer firewall, simply blocking or allowing content based upon rule evaluation. To perform all three of the web threat mitigation methods given at the start of this section modern secure web gateways act as application level proxies (or inline proxies) and thus can alter content on the fly to insert user notifications or sanitise content.

A secure web gateway is thus, at its core, a web proxy that modifies content that passes through it. We used the open source Squid [Squid, 2010] proxy due to its popularity, project maturity, and wide deployment. There are several main technological methods used for content modification in squid: altering the code to insert the required behaviour and recompiling, Internet Content Adaptation Protocol (ICAP), [Elson & Cerpa, 2003], Client Streams [Squid Wiki, 2010], eCAP [ECAP, 2010], and modified squid configuration settings. Each of these content adaptation methods has advantages and disadvantages, however ICAP is the most standardised. Furthermore, Comcast (Figure 1) has deployed a trial implementation for user notification using Squid and the Greasyspoon ICAP server [Chung, 2010], and we built upon a similar model adding the other two security operations of content blocking and content sanitisation.

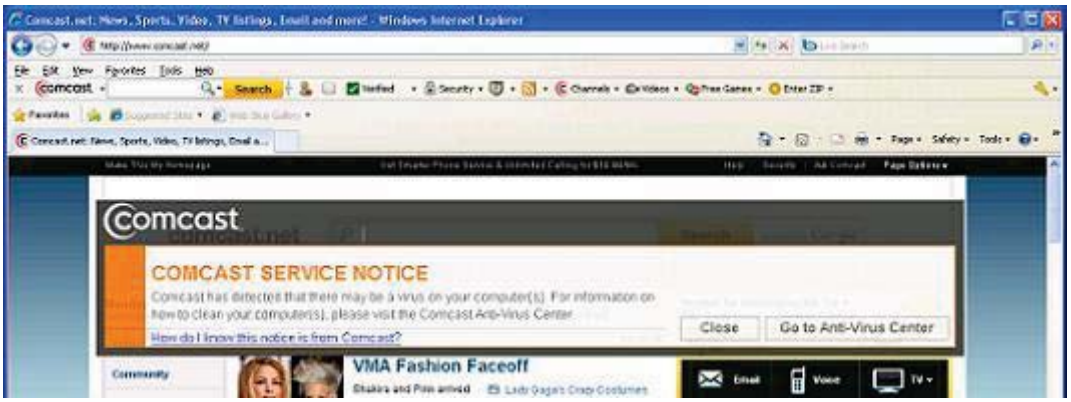


Figure 1: Comcast's Constant guard notification [Comcast, 2010]

Greasyspoon is a Java based open source ICAP server that runs individual script modules which can be enabled, disabled, and edited seamlessly while the system is running. The scripts can be written in many different languages (as Greasyspoon uses Java's JSR223), however our scripts were written in Java to remove the impedance mismatch between different languages. We discuss this system developed in the next section.

SYSTEM DESIGN

All of the components we used are open source, with the core components being the Squid web proxy server and the Greasyspoon ICAP server and additional components being used for specific functionality including a MySQL database server, a ClamAV server, and a RADIUS authentication server.

In the proof of concept implementation (Figure 2) the servers were spread across several identical physical machines on a switched gigabit network so that network communications between them could be observed. Netperf gave speeds of 111.66 MB/s between switched clients, which was more than adequate for our tests. There will be latency however, which will add delay and resource overhead, so in a production environment many of the servers would be physically consolidated, permitting faster communication between different system components.

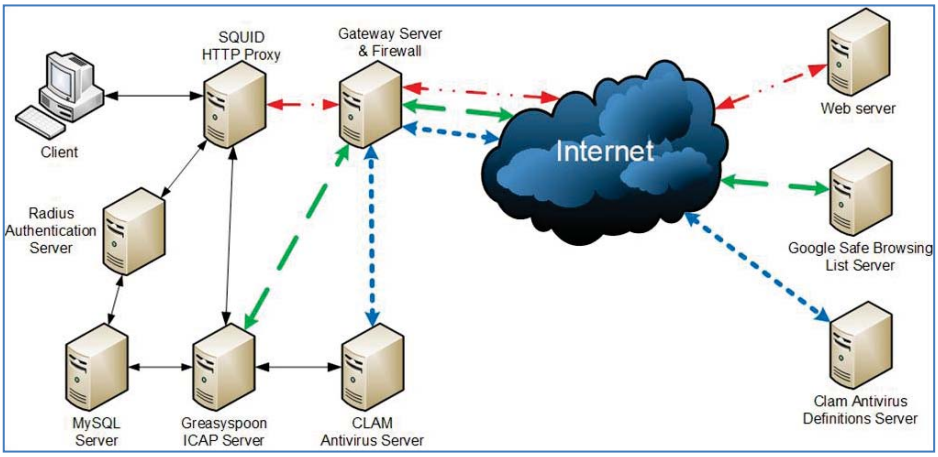


Figure 2: The logical layout of the developed system

Using this system, the web client is set to use the web browser in the proxy settings so that all web traffic goes through the squid proxy[4] and is processed by the Greasyspoon ICAP server[5]. Squid has the caching functionality disabled at present, to ensure that caching does not cause problems for scripts where performance optimisation in the subject of future research). Non-http traffic does not use the web proxy however, and goes through the gateway directly. To perform security web content modification, scripts were set up on the server to perform the following types of operation (Table 1):

Table 1: Security Content Modification Types

General Content Modification Type	**Example Security Use**
Request Header Injection	Inserting *Cache-Control: no-cache* To attempt to bypass poisoned caches
Request Header inspection and alteration	IP, referrer, or User-Agent Anonymisation
Request URL redirection	Redirection of risky document types to online viewers (eg PDF's to Google Docs viewer)
Request to response modification	Instead of loading a blacklisted site, respond with a warning page.
Response Content Insertion (Static Conditional)	Warning entire userbase (ie phishing warnings)
Response Content Insertion (Value Lookup)	User notification of security situations (ie Comcast)
Response Inspection and alteration	Deactivation of inline scripts or media (ie flash)
Response replacement	Replacing virus infected files with warning pages.

From these categories we developed several scripts for use in the verification of the system (Table 2):

Table 2: Script functionality developed and tested

Script Type	**Action**
Request URL Redirection	Redirection of requests for certain risky file types (such as pdf) to an online viewer (See Figure 3 below)

[4] It is possible to set the Squid proxy up in transparent mode so that web traffic is forced to use the proxy (a production environment would probably do this), and we did test this functionality, but the manual setting of the proxy was retained to allow the comparative testing we undertake further on.

[5] Initially Greasyspoon required a configuration modification and source code modification to enable direct binary access to content, however after communication with the project leader (Karel Mittig) and sending our changes this functionality was integrated into the next Greasyspoon release (albeit in a different way).

Request to response modification	Site blocking
Request header alteration	Browser User-Agent anonymisation
Response content insertion (Static Conditional)	Global userbase messaging from database
Response Content Insertion (Value Lookup)	User specific messaging from database (by verifying user either through the proxy with radius, and by IP address)
Response Inspection and alteration	Keyword based blocking / alteration
Response Inspection and Replacement (Binary-level)	File type blocking (MIME, file signature)
Response Replacement	Antivirus scanning of responses.
Request / Response (Pair)	Browser user agent detection and notification

```
//-----------------------------------------------------------------
// ==ServerScript==
// @name          onlinedocumentviewer
// @status on
// @description Redirects request URL's with certains endings to the google viewer, preventing them loading in the browser env
// @include       .*pdf
// @exclude
// @timeout    300
// ==/ServerScript==
// ---------------
public void main(HttpMessage httpMessage){
   String[] extensions = {"pdf","ppt","doc","docx"};
   String url = httpMessage.getUrl();
   boolean redirect = false;
   String referrer = httpMessage.getRequestHeader("Referer");

   // The pdf detection is a little weak, ideally should be partnered with a signature scanner script for response
   //I.E. look for "%PDF-" at the start of the file or the correct MIME type
   //We check for the google referrer to enable file download should it be desired.
   if(!url.contains("docs.google.com") && !referrer.contains("docs.google.com") ){
      for(int i = 0;i < extensions.length;i++){
         if(VERBOSEMODE) System.out.println("Checking " + url + " for: " + extensions [i]);
         if(url.toLowerCase().endsWith(extensions[i]))
            {
            redirect = true;
            break;
            }
      }
      if (redirect == true){
          String encodedPageUrl = java.net.URLEncoder.encode(url);
          String redirectUrl = "http://docs.google.com/viewer?url=" + encodedPageUrl ;
          String redirectHeader = "HTTP/1.0 302 Found" + "\r\n" + "LOCATION:" + redirectUrl + "\r\n";
          httpMessage.setHeaders(redirectHeader);
      }
   }
}
```

Figure 3: SCRIPT FOR REQUEST REDIRECTION

TESTBED AND TESTING METHODOLOGY

Two main aspects were considered to be important in order to assess the usefulness of any network security service such as this, viz. effectiveness against threats, and performance impact upon users.

While there are no security operations we attempt to perform that are in any way novel or controversial, we chose to test the effectiveness of our system against real world attacks nonetheless. All simple scripts (such as those for user notification) were verified manually due to their performing very simple operations.

We adopted a very different approach with the automated security operation scripts however, choosing to test them by using comparative high-interaction client honeypots on live URLs from two data sets. URLs were used from known malicious sets, and automatically gathered URLs from online trends at the time of testing (to ensure active and current threats).

The effectiveness and performance testing will each be discussed in more detail, but an overview of the logical architecture of the testbed is shown in Figure 4 below. Note that only one of the effectiveness and performance testing were connected at any point.

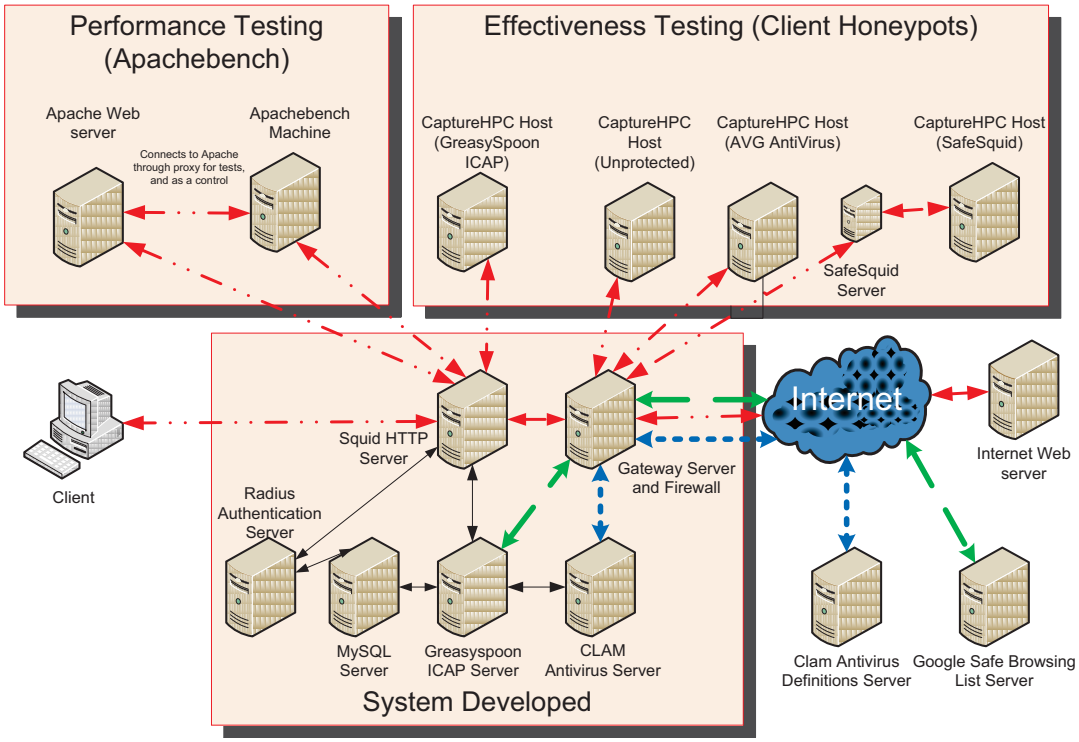


Figure 4: Architecture of the testbed

EFFECTIVENESS TESTING

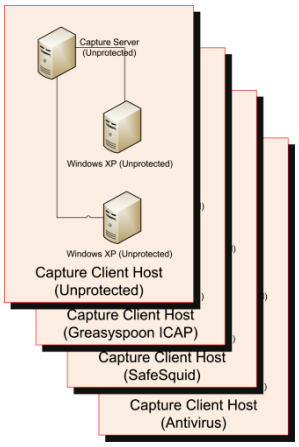
We undertook testing to obtain quantitative data regarding the effectiveness of our system against attacks by running it alongside existing standard protection measures and comparing outcomes. Thus, we undertook identical effectiveness testing in parallel using an unprotected system, a system running a consumer antivirus product, a system running a competing secure proxy solution, and the system we devised. Testing was undertaken using two sets of URLs, one set of "known bad" (3 day old) data from a blacklist site, and the second "real world" set. The real world set was obtained from Twitter using a script that searches the Twitter site for the current Twitter trends and extracts corresponding URLs. This resulted in many of the known bad sites being offline, and a significant portion of the real world set being shortened URLs such as bit.ly[6].

The automation of effectiveness testing was undertaken using the Capture-HPC [Seifert, 2007][HPC, 2010] High Interaction Client Honeypot (HICH) system. The Capture-HPC system is designed to detect malicious websites through the use of a standard client operating system and software running in virtual machines. One Capture Server can operate multiple virtual machines (called "Capture clients") across multiple physical machines, but our system uses only one server running two clients on the same hardware due to the server only functioning correctly on one TCP listening port, i.e. only one server can run per physical machine.

In brief, Capture assesses the security of websites by sending each client (in a known state) to a group of pages, and checking for non-whitelisted events involving the alteration of files, running processes, network connections, and the Windows registry. Should an unauthorised event occur, then relevant details are logged and sent to the controlling server followed by reversion of the client virtual machine to the known "clean" state virtual machine snapshot.

[6] bit.ly involves using an HTTP redirect on a domain name that is short to link to a website which has a long URL, i.e. shorturl=http://www.reallylongsitenamesite.com/subfolder/subsubfolder/pagename.html. These are very popular on social network sites like Twitter where there are very tight character limits.

We ran a single host machine for each protection measure, with each machine running one Capture server, and two clients. Capture HPC version 3.01 and VMware server 1.09 were used on the host (Figure 5).



Client Virtual Machine Setup	
Operating system	**Windows XP SP2**
Web browser	• Internet explorer 6 (Unpatched since sp2) • With Asian language packs installed
Browser plugins	• Flash 10 (various versions) • Java (From Java 6 u 16) • Acrobat reader 8.1.1 • Quicktime 7 (various versions)
Other software	• Java 6 update 16 • Acrobat Reader 8.1.1 • Quicktime
Addit Config	• Clickoff set to automatically run any file download dialogs offered, and bypass internet explorer warning dialogs. This was done to simulate a naive user.

Figure 5: Setup of Client Virtual Machine

Using this setup on modest hardware (four systems of Core2 duo 2.1GHz with 2GB RAM) we were able to check around 10,000 URLs/day for the real world set and 15-20,000 URLs/day for the known bad dataset due to the much higher rate of DNS errors. At this stage the whitelisting is problematic though, with around 98% of the malicious events in the real world set being due to false positives, and around 60% in the "known bad" dataset. This results in a very high false positive rate, and at this stage manual inspection seems to be the quickest way to verify. We suspect that this is due to the complex array of software installed in the client, and interactions with the antivirus software file hooking make the problem even more noticeable on the antivirus clients.

PERFORMANCE TESTING

Performance testing was also deemed important, and was tested across several variables (Table 3). Tests were undertaken using [Apachebench, 2010] from a system on the same network. While this simplification of only one webserver will cause some loss of real world results, it should enable the determination of system settings and parameters for optimal real world use.

Table 3: Variables used in testing

Variable	Values Used
File Size	1KB, 10KB, 100KB, 1MB, 10MB
File Type	Html (lorem ipsum), binary (pregenerated from /dev/urandom)
Concurrency	1, 10, 100, 200, 400
Request Load	166/s, 1667,s
All above	-

These variable values were chosen after testing the system to failure, and a value around the failure point was chosen for the top end of each variable. For test cases, In addition to each script class were added three additional control cases, no proxy, proxy and no icap, and proxy with icap but no scripts enabled. This enabled us to remove the effects of additional network overhead due to the operation of proxies and icap. For example a proxy will tend to require a full transmission of the data twice, and ICAP at least three times to go between the different servers.

Table 4: Test cases used in test bed architecture

Testcase	Notes
No proxy (thus no ICAP)	CONTROL (needs results adjustment to allow for reduced network traffic)
Proxy, no ICAP	CONTROL
Proxy, ICAP, no scripts	CONTROL
Request Header insertion	
Request header inspection & alteration	
Request URL Conditional Change	
Request to Response Modification	
Response Content Insertion (Static Conditional)	To test content insertion delay
Response Content Insertion (Continual DB lookup)	To test database operations in scripts
Response Content Insertion (Caching DB lookup)	To test effectiveness of caching using the Java shared cache
Response Antivirus Inspection	To test performance effects of sending entire content body from icap through network also
ICAP with One of each type above	

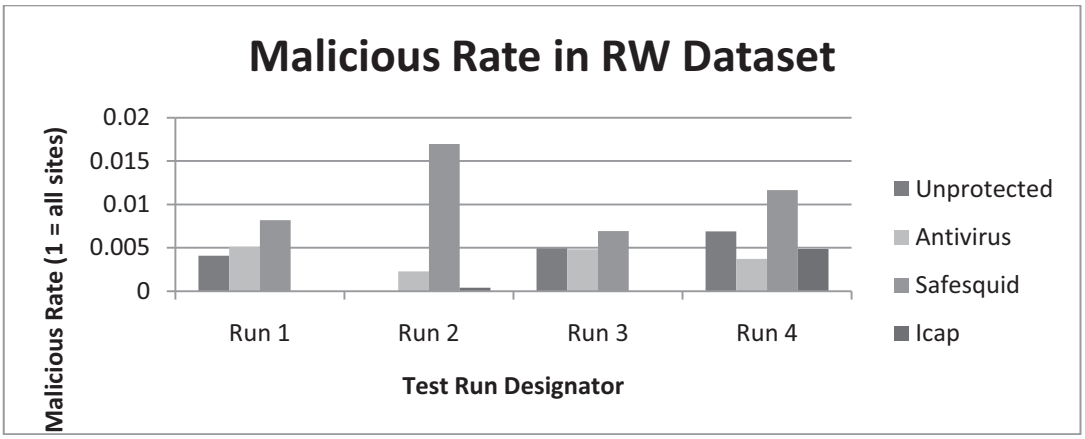
The combination of these variables with the 12 test cases given below gave a total of around 500 tests each with tens of thousands of data points. The result of this is that we have a very good picture of the behaviour of our system under various combinations of workloads and situations.

RESULTS

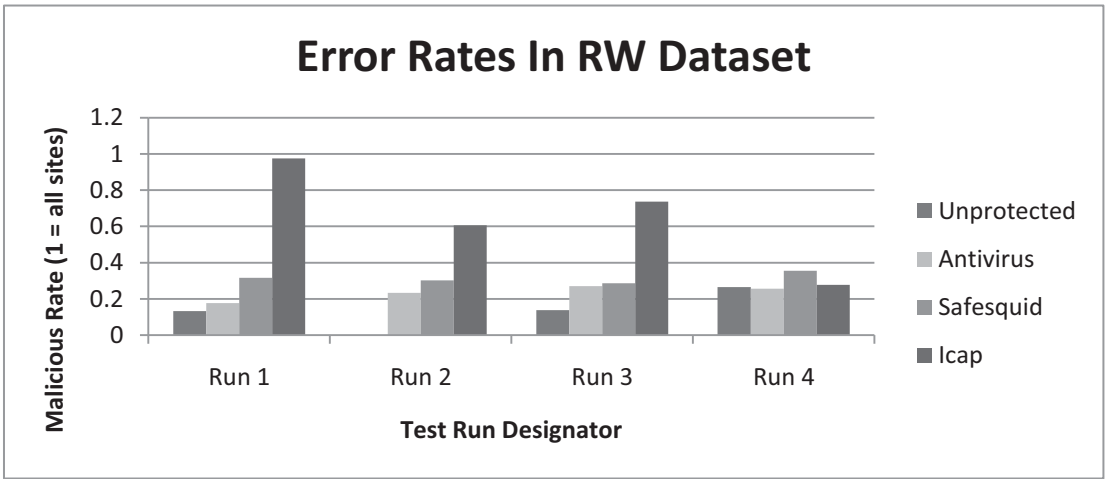
The following section describes a sample of the results obtained but is only indicative of a full analysis.

Effectiveness

Sample results from early testing are illustrated below. We have provided sample data from some of our early test runs to indicate how the data appears, and what it can indicate. The following two graphs show the malicious rate and error rates in "Real World" (RW) datasets against the malicious rate for a number of runs.

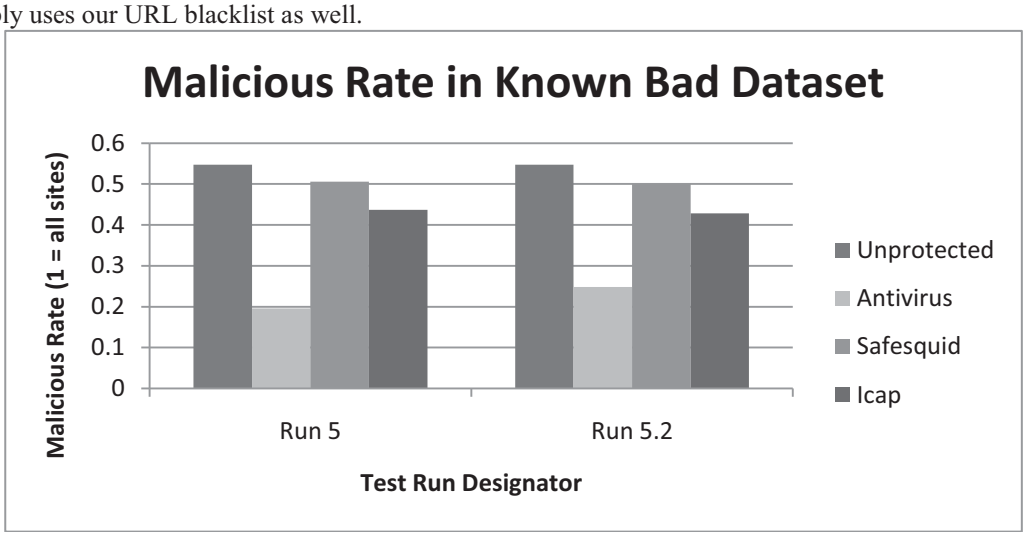


As we proceed further into testing we are gradually improving the whitelist used by capture, and improving some of the scripts to deal with errors better. We have given the error rates in the graph below as well, as they make these results easier to understand.



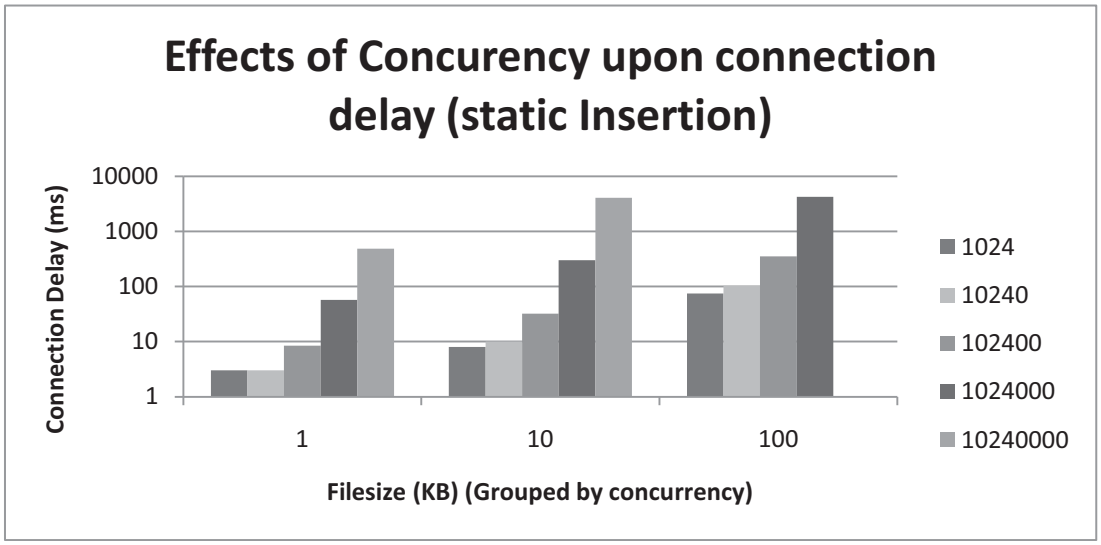
The first three runs are to be discarded, as they do not have close error rates between the different protection measures. But what is notable is that for run 4 with comparable error rates between the measures the results look closer to one that would be expected, with the exception of the competing proxy based solution having a higher malicious rate than unprotected. We are still improving our system though, and as our error rates get lower and closer the results are becoming more interesting.

What is particularly interesting is that in our "known bad" (KB) dataset the antivirus has a noticeably lower malicious rate than the other measures. We suspect this is due to the signature-based scanning used by most antivirus solutions, and it also possibly uses our URL blacklist as well.



Performance

Different scripts have massively different performance characteristics, as would be expected given their different overheads. Concurrency increases connection throughput per second, but increases latency per connection. This achieves a peak before it hits the connection's throughput limit. The exact balance of this would require tuning per application, or perhaps content type. A small application would desire less concurrency for less latency, but a larger organisation would likely require more raw throughput, and thus higher concurrency.



In general however the setup needs performance optimisation before real world deployment would be possible. This would require physical consolidation, faster hardware, and likely deployment on an OS with more than 1024 file descriptors[7].

CONCLUSION AND FURTHER WORK

Overall, preliminary results seem to show promise for real world application of this system, however this bears further analysis of the data we are gathering. There is much further work that can be done to expand upon this project, both enhancements of the existing and extensions of functionality.

Improvements are needed with the performance aspects, particularly with database reads and concurrency, but also with investigation into proxy caching and its effects on the system security and performance. Physical consolidation should offer significant improvements, but may make load balancing more difficult. Additional investigation is warranted into using the same method Comcast did for notifications – using a web server for serving the notifications. Testing should also be undertaken with a more modern and less contrived honeypot client, as most people do not deliberately install vulnerable software versions!

Expansions to this system could include functionality such as https inspection (via SSLBump), full signature scanning (via SNORT signatures), data loss prevention, and more advanced versions of some of the scripts (such as cross site scripting detection). We foresee the ideal solution developed from this to be a server-based security solution that is administered by the user using controls injected into the browser stream.

The testing developed for this project could also warrant development into a more broad security testing methodology.

REFERENCES

Apachebench - apache http server benchmarking tool - apache http server. <http://httpd.apache.org/docs/2.0/programs/ab.html> [Accessed 13 September 2010]

Beauvisage, T. Computer usage in daily life. Proceedings of the 27th international conference on Human factors in computing systems - CHI '09 575(2009).doi:10.1145/1518701.1518791

Capture-HPC Webpage. <https://projects.honeynet.org/capture-hpc>, 2010. [Accessed 13 September 2010]

Chung, C. et al. Example of an ISP Web Notification System. Internet Engineering Task Force Internet Draft 1-22 (2010).

Comcast Constant Guard Notification, 2010, <http://security.comcast.net/img/content/InBrowserNotice.jpg> [Accessed 13 September 2010]

[7] During testing it was found that Ubuntu ran out of file descriptors and started dropping connections

Danchev, D. Scareware pops-up at FoxNews | ZDNet. ZDnet.com - Zero Day, <http://www.zdnet.com/blog/security/scareware-pops-up-at-foxnews/3140?p=3140> [Accessed 13 September, 2010]

eCAP, <http://www.e-cap.org/Home> [Accessed 13 September, 2010]

Elson, J. & Cerpa, A. RFC3507: Internet Content Adaptation Protocol (ICAP). RFC Editor United States (2003), <http://portal.acm.org/citation.cfm?id=RFC3507>

Keats, S. Mapping the Mal Web, Revisited. McAfee SiteAdvisor 1-26(2009), <http://www.siteadvisor.ca/studies/map_malweb_jun2008.pdf>

McAfee WhitePages squashes ad networks after finding malware |. MX Logic, <http://www.mxlogic.com/securitynews/web-security/whitepages-squashes-ad-networks-after-finding-malware515.cfm> [Accessed 13 September, 2010]

New York Times Websit Note to Readers - NYTimes.com. (2009). <https://www.nytimes.com/2009/09/13/business/media/13note.html?_r=2&hp>

Poulsen, K. Cybercrooks Trick Gawker Into Serving Malware-Laced Ad. Wired.com - Threat Level (2009), <http://www.wired.com/threatlevel/2009/10/gawker/>

Seifert, C. et al. Capture – A behavioural analysis tool for applications and documents. Digital Investigation 4, 23-30(2007).

Squid : Optimising Web Delivery, 2010, <http://www.squid-cache.org/>

Squid Wiki, ProgrammingGuide/ClientStreams - Squid Web Proxy Wiki. <http://wiki.squid-cache.org/ProgrammingGuide/ClientStreams> [Accessed 13 September 2010]