Protecting the Web with Transparent Proof-of-Work

Ed Kaiser, Wu-chang Feng

Portland State University

Supported by:

intel
NSF
Motivation

• Unwanted web traffic is everywhere
  – Denial of Service
  – Comment spam
  – Click fraud
  – Ticket robots
  – Fake web account signup
  – Duplicate on-line voting

• Observation
  – Most attacks are automated
CAPTCHAs to the rescue!

• Use a hard AI problem for security
  – Force users to solve a problem that is hard for a computer, but easy for a human
  – Turing test that does not require special client software

• Widely used
  – Google
  – Microsoft Live/Passport/Hotmail
  – Yahoo!
  – phpBB
CAPTCHA Problem #1

• User-interface problem
  – Inaccessible to visually impaired
  – Some inaccessible to normal users
  – Designed with several attempts in mind
    • frustrating, annoying, aesthetically unappealing experience
    • not suitable for frequent transactions
CAPTCHA Problem #2

- Adversaries solving the hard AI problem
  - Improvements to OCR erodes effectiveness
  - Examples
    - Yahoo! broken 1/2008
    - Windows Live/Passport, Google reported broken 2/2008
    - PWNtcha CAPTCHA solving library

<table>
<thead>
<tr>
<th>Origin</th>
<th>Samples</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>linuxfr.org</td>
<td>![Sample Image]</td>
<td>100%</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>![Sample Image]</td>
<td>99%</td>
</tr>
<tr>
<td>Paypal</td>
<td>![Sample Image]</td>
<td>88%</td>
</tr>
<tr>
<td>phpBB</td>
<td>![Sample Image]</td>
<td>97%</td>
</tr>
<tr>
<td>SCode and derivatives</td>
<td>![Sample Image]</td>
<td>100%</td>
</tr>
<tr>
<td>Slashdot</td>
<td>![Sample Image]</td>
<td>89%</td>
</tr>
</tbody>
</table>
CAPTCHA Problem #3

- Economics broken
  - Fixed workload priced at 10 seconds of human time
    - Outsourced for under 1¢ per CAPTCHA

- CAPTCHA pricing does not work
  - When adversary resources are vastly greater than legitimate ones
  - When value of what is being protected is more than 1¢
Need a variable workload to price out adversaries!
Proof-of-Work (PoW)

• Alternative to CAPTCHA
  – Clients solve a computational puzzle to get access

• Addresses CAPTCHA problems
  – No user interface issues
  – Adversary must solve a hard cryptographic problem
  – Adjustable difficulty that treats CPU cycles as currency
But…

- Landscape littered with unused PoW schemes!
  - Hash cash, TLS puzzles, TCP puzzles
  - IP puzzles, Public puzzles (two of our own stinkers)

- Why?
  - Introduces a big problem CAPTCHA does not
  - Forces changes to network protocols and software
  - Client must install PoW software to participate
Our approach: \texttt{mod\_kaPoW}

- **Provide benefits of PoW without changes to client**
  - Apache module
    - Dynamically embeds PoW with client-specific difficulty into URLs
    - Attaches JavaScript solver for client to run
    - Verifies subsequent solutions
  - Client browser
    - Runs JavaScript solver to calculate answers
    - Attaches answers to subsequent URL requests
  - No protocol changes
  - No web browser changes
  - No web content changes
mod_kaPoW architecture
mod_kaPoW puzzle

• Based on targeted hash reversal
  Global Internet 2007

• Server attaches puzzle to embedded links
  – $N_c =$ client-specific server-generated nonce
  – $D_c =$ client-specific server-assigned difficulty

• Client JavaScript solver finds $A$ such that
  
  $\text{SHA1}(N_c \ || \ URL \ || \ A) = 0 \mod D_c$

  – Brute-force search requiring $D_c$ SHA1 hashes on average to find

  – Attaches $N_c$, $D_c$, and $A$ to URL to access content
Example

• Original content on disk

```html
<HEAD>
  <TITLE>kaPoW!</TITLE>
</HEAD>

<BODY>
  <A HREF="protect_me.html">Protected Link</A>
</BODY>
```

• Content after Apache embedding of PoW

```html
<HEAD>
  <SCRIPT TYPE='text/javascript' SRC="/kaPoW.js" Nc=F2DCFC86 Dc=200"></SCRIPT>
  <TITLE>kaPoW!</TITLE>
</HEAD>

<BODY>
  <A HREF="protect_me.html">Protected Link</A>
</BODY>
```

– JavaScript solver `kaPoW.js`
  • Registers “onLoad” and “onClick” event handlers
  • Implements SHA1 to solve PoWs of URLs given puzzle parameters
    – “onLoad” for embedded images
    – “onClick” for embedded links
Demo

Overhead

- Negligible for dynamic page
- Small fixed amount for static page
- Fast verification and rejection
Thwarting DoS

- Simple experiment
  - Good client at 1 request per second
  - 6 flooding adversaries attack at 35 second mark
  - Counting Bloom Filter used to track usage and set difficulty
What next?

• Towards a computational approach for protecting Internet applications

• Building applications around kaPoW
  – Treat CPU cycles as currency and create virtual markets
  – Use cycles to create incentives for proper behavior
  – Force adversaries (spammers, ticket brokers, hackers) to “pay” for access
    • A tax paid to Intel!
Tackling comment spam

• Content-based difficulties
  – Force “spammy” comments to use a large amount of cycles
  – Send posts through SpamAssassin and use its score to determine puzzle difficulty

• Weighted voting
  – Allow users to “vote” on comments with their CPU cycles
  – Promote comments with the most committed cycles

• Community-assisted pricing
  – Allow users police the price for posting for each other based on prior posts
  – Use “karma” (Slashdot) to determine CPU cycles a particular user needs to post
Tackling click fraud

• Increase click costs on suspected fraud
  – Apply credit-card fraud techniques to detect possible fraud
  – Increase CPU tax on ad click-throughs that are suspicious
    • Use prior history of clicks to prevent Auction Experts employees from “clicking-through” Google ads
Tackling ticket robots

• Increase cost of “purchase” link geographically
  – Use MaxMind/GeoIP to determine where clicks originate
  – Increase costs on those far away
  – Forces ticket robots to be located in each city
    • Much better economics than $0.01 CAPTCHAs!
Roadmap

• Adding to LAMP stacks
  – Linux, Apache, MySQL, PHP/Perl
  – Allowing applications to control difficulty
  – phpBB, WordPress, Twiki, Drupal, guestbooks

• Using with CAPTCHA
  – Frequent transactions protected with kaPoW
  – Infrequent transactions protected by both
A brief plug on AMT work

• **CS 576: Detecting Cheating in Online Games**
  – Repeating last year’s successful offering
  – Using Intel’s AMT as an undetectable debugger
  – What exploits used by cheat software could be reliably measured by the AMT?

• **NSF FIND, GENI**
  – Clean-slate design of the Internet
  – Building Future Networks Around Ubiquitous Use of AMTs
    • Trusted Third Parties make many security protocols easy
    • Can TPMs acting as TTPs fix problems in network protocol design?
    • An interesting academic exercise (for now)
Questions?

http://kapow.cs.pdx.edu
Extra slides
Addressing economics

• How do you construct a pricing system that works?
  – What is the cost of unattended (idle) CPU cycles?
  – Can costs be controlled to create sufficient disincentives for botnets of 20,000 idle machines?
  – How much is it worth to keep bots hidden?
  – How do you cope with price limits to legitimate users?