Designing and implementing secure web browsers

or

How to keep your cores busy for two seconds at a time

Chris Grier, Shuo Tang, Samuel T. King
Motivation

• Browser most commonly used application today
• Browsers are an application platform
  – Email, banking, investing, shopping, television, and more!

• Browsers are plagued with vulnerabilities
  – Internet Explorer: 57 vulnerabilities
  – Mozilla/Firefox: 122 vulnerabilities
  – Safari + Opera: 66 vulnerabilities

• Studies from Microsoft, Google, and University of Washington show web browser is attacker target
Anatomy of a browser attack

• What could a browser attack look like?
Designing and implementing malicious hardware

Samuel T. King, Joseph Tuex, Anthony Cozzie, Chris Grier, Wei Wang Jiang, and Yuanyuan Zhou

University of Illinois at Urbana Champaign, Urbana, IL 61801

Abstract
Hidden malicious circuits provide an attacker with a stealthy attack vector. As they occupy a layer below the entire software stack, malicious circuits can bypass traditional defensive techniques. Yet current work on trojan circuits considers only simple attacks against the hardware itself, and straightforward defenses. More complex designs that attack the software are unexplored, as are the countermeasures an attacker may take to bypass proposed defenses.

We present the design and implementation of Illinois Malicious Processors (IMPs). There is a substantial design space in malicious circuitry; we show that an attacker, rather than designing one specific attack, can instead design hardware to support attacks. Such flexible hardware allows powerful, general purpose attacks, while remaining surprisingly low in the amount of additional hardware. We show two such hardware designs, and implement them in a real system. Further, we show three powerful attacks using this hardware, including a login backdoor that gives an attacker complete and high-level access to the machine. This login attack requires only 1341 additional gates; gates that can be used for other attacks as well. Malicious processors are more practical, more flexible, and harder to detect than an imp.
The OP Browser

• Goal: build a secure web browser

• Provide an architecture for secure web browsing
  – Maintain security guarantees even when compromised
  – Integrate plugin policy into overall browser policy

• Use OS techniques, formal methods
  – Partition browser
  – Expose communication
  – Reason with formal methods
  – Analyze attacks
Gazelle

- Goal: improved display security
- More fine-grained isolation
  - Enables novel display security policies
- Trade off compatibility for security
OP2

• Culmination of recent work in secure browsers
  – Based on OP
  – Borrows ideas from Chrome and Gazelle

• Surprising preliminary performance results
  – Modifications for security improved performance
Outline

- OP browser design
  - Using formal methods to verify invariants
  - Performance

- OP2 and Gazelle
  - Display security

- **OP2 performance**

- Other research from my group
Threat Model

• Threat model: the attacker is targeting the browser and has complete control over content being served in the web page
OP design

• Decompose into browser subsystems
  – Web page instance further divided

• Use message passing
  – All messages through browser kernel

• Dedicated subsystems for OS operations

• Host OS sandboxing
Design enables security

• Partitioning and constrained communication enable new security mechanisms
  – Clean separation of browser functionality and security

• Policy
  – Easier to reason about current policies
  – Novel policies including for plugin security

• Formal methods

• Forensics
Use of formal methods

• Model using Maude
• Attack modeled by sending arbitrary messages

• Check SOP policy

• URL bar = URL loaded with compromise
  – Model checking revealed paths to bad state
  – Attacker could send out of order messages

• Use to drive development
  – Fix bugs, update model, re-check
Implementation

• Use KHTML as rendering engine
• Rhino for JavaScript
• Use Java where it makes sense
  – C++ for browser kernel, only about 1000 LOC
Performance (circa 2007)

• Load latencies do not impact usability
The Gazelle Web Browser
Helen J. Wang, Chris Grier, Alex Moshchuk,
Sam King, Piali Choudhury, and Herman Venter

Microsoft Research, UIUC, U. Washington
Gazelle architecture

- Per-origin processes
  - Sandboxed

- Gazelle system calls for accessing resources
  - handled by browser kernel

- Browser instance is libc for web
  - Gazelle syscalls built-in
  - HTML handling
  - JS execution

- Browser Kernel
  - Access to system resources
  - Enforces all security policy
Display security in Gazelle

• Goal: Provide strong isolation between rendered content

• Compose content from many different services securely

• Not as clear decisions as traditional OS display
  – Cross principal content inherent in rendering on web

• Difficult cases can raise policy questions
  – Frames can be transparent
  – Images under text
  – Layers in CSS
What is display isolation?
Redressing Flash

All your mics are belong to us!

Do you allow AJAX?
AJAX will improve your user experience!

UI Redress attack against Flash

• http://www.flickr.com/photos/24967759@N00/2924995732/
A website from adobe...

Privacy pop-up question

Why do I need to answer this question?
What happens if I select Allow?
What happens if I select Deny?
Do I have to answer this question every time I run an application from this website?
How can I display this question again?

Why do I need to answer this question?
The application running in Flash Player has requested access to the camera and/or microphone available on your computer, from now until the application ends. Note that it is the person or company that has created the application you
Do you allow AJAX?
AJAX will improve your user experience!

UI Redress attack against Flash

- http://www.flickr.com/photos/24967759@N00/2924995732/
Delegate once policy

• Delegate once policy
  – Delegate, screen space is lost for duration of the page
  – Cannot draw over or outside of delegated space

• Deviate from standards for improved security
• Prevents drawing over cross-origin content
• Helps (but doesn’t eliminate) “UI redressing”

• Going to break certain things (menus, move ads, full scrn)
Display isolation mechanisms

• Enforce display policy in browser kernel
• Cross-domain iframes and plugins isolated
  – Rendered in separate processes

• From paper, unclear if this will be practical
  – Significant overhead for nytimes.com
OP2: making secure browsers more practical

Shuo Tang, Chris Grier, Sam King

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
• Based on original OP browser
• From Chrome: combine JS and HTML rend.
  – Content sniffing algorithm from Barth et al.
• From Gazelle: display security mech and policy
OP2 implementation

• Implemented using WebKit and Qt
  – Linux and Mac version, results for Linux
  – Entire browser now in C++ to use Qt
  – Browser kernel about 1500 LOC

• Subtle diffs between OP2, Gazelle, and Chrome
  – OP2 reuse existing components when possible
  – Try to keep things as simple as possible
    • Fewer features
  – Less worried about compatibility
    • Try to maintain compat policies when possible
OP2 performance

• Performance experiments
  – Page load latency times, visit 10 times after warm
  – Compare vs latest version of Arora
    • Also uses WebKit and Qt, single process architecture

• Experimental setup
  – 2.66 GHz Core 2 Duo
  – 8 GB of RAM
  – Connected to school network
Simple optimizations

• Pre-create web page instance processes
• Overlap window mgr ops w page loading
OP2 no optimizations
OP2 with process pre-creation
OP2 with parallel win mgr operations
Arora

time in seconds

sites: google.com, cs.illinois.edu, bing.com, en.wikipedia.org, sfbay.craigslist.org, nytimes.com
Adding parallelism in OP2

**Flowchart:**

1. Cont processing
2. Arora / WebKit
   - Fetch URL
   - Setup headers
   - Check for cache hit
   - ...
   - Async syscall

**User space**

**Kernel space**
Adding parallelism in OP2

- OP2 Web Page Instance (WebKit)
  - Fetch URL
  - Async syscall

- OP2 Net Process
  - Fetch URL
  - Setup headers
  - Check for cache hit
  - ...
Comparison of time in seconds for different optimizations:

- **OP2 no optimizations**
- **OP2 with process pre-creation**
- **OP2 with parallel win mgr operations**
- **Arora**

Sites included:
- google.com
- cs.illinois.edu
- bing.com
- en.wikipedia.org
- sfbay.craigslist.org
- nytimes.com
Process cache optimization

• WebKit built assuming process reuse
  – Cache web object in memory
• Starting from a fresh state fundamental to OP
  – Security purposes

• Solution: cache old web page instances
  – Hits only when we visit the exact same URL
  – Minimize amount of state that could be leaked
Display isolation mechanisms

• Have a fully optimized OP2 browser
  – Determine if display isolation could be practical

• Put cross origin iframes in separate processes
  – Done for security reasons, easier to label
  – Can enforce display policies in OP2 browser kern.
Lessons learned from OP2 eval

• Changes for security improved performance
  – Usually shoot for 100% overhead or less
    • 25% don’t even have to explain

• Huge opportunities for performance gains
  – Performance optimizations for architecture
  – Accidentally improved performance
Related Work

• New architectures
  – Using VMMs: Tahoma [Oakland ‘06]
  – File system focused: *Building a secure web browser* [FREENIX ‘01]
  – Process based: *Architectural principles for safe web programs* [HOTNETS ‘07]

• Securing existing applications and new abstractions
  – Javascript: Browsershield, JS Instrumentation [OSDI ‘06, POPL ’07]
  – Privacy: Safecache/Safehistory [WWW ‘06]
  – SOP: locked SOP, Script accenting [CCS ‘07]

• Formal methods
  – UI invariants: *A systematic approach to uncover security flaws in gui logic* [Oakland ‘07]

• Other attacks
Conclusions

- Treat browser like an OS, more secure
- OP and OP2 improve security
  - OP2 also improves performance

- A step towards preventing, containing, and recovering from browser-based attacks
- A step towards a parallel web browser
Questions?

• (Note: this is not the end of my talk yet)
Untrusted computing base: defending against malicious hardware

Matt Hicks, Matt Finnicum, Sam King
University of Illinois

Milo M.K. Martin and Jonathan Smith
University of Pennsylvania
Building secure systems

• We make assumptions when designing secure systems
• Break secure system, break assumptions
  – E.g., look for crypto keys in memory
• People assume hardware is correct

• What if we break this assumption?
Malicious hardware

• Is it possible to modify design of processors?

• Implementing hardware is difficult
• Implementing HW-based attacks is easy!
  – Small hardware level *footholds*
  – Execute high-level high-value attacks WITHOUT exploiting any software bugs
Defenses

• Based on insights from foothold devel.
• Analyze circuit at design time
• Highlight potentially malicious circuits

• Hope to have results soon
Deterministic replay

- Record execution, reproduce arbitrary past states
  - Debugging, fault tolerance, security, etc.
- SW-only replay flexible, slow for MP machines
- HW-only fast, record entire machine
  - Less flexible for current uses
- Combine HW and SW replay
  - Naïve approach does not work
  - Subtle and fundamental issues
- Capo: HW/SW interface and abstractions for record and replay
- Paper in ASPLOS ‘09
  - Joint work with Pablo Montesinos, Matt Hicks, and Josep Torrellas
Digging for Data Structures

Anthony Cozzie, Frank Stratton, Hui Xue, Sam King
University of Illinois at Urbana-Champaign
Data Structure based Antivirus

• Detect programs based on their data structs
• Convert seemingly random bytes of program memory into data structures automatically
  – Mark each word as an int, pointer, string, etc.
  – Use Bayesian classifier – see paper for details
• Two programs with same data structures are likely the same program
  – Worked for three modern botnets
• Presented at OSDI ‘08
Other projects

• Automatic fault recovery
  – Paper in ASPLOS ‘09
  – Joint work with Andrew Lenharth and Vikram Adve

• More secure web browser work
Questions?

• (Note: this is the end of the talk)
Replicate portions of the OS

• Extracts parts of OS needed for web client sec
  – Custom labeling and access control system
  – RPC / message passing layer
  – Window manager (limited extent)
Assumptions about OS

• Process-level isolation
  – Memory protection
  – well-known IPC mechanisms

• System-level sandboxing
  – Isolate processes from system resources
  – Restrict system call capabilities

• Resource management
  – Create processes, message forwarding and naming
  – Network, disk, screen
## Differences between OP, Chrome, Gazelle and OP2

<table>
<thead>
<tr>
<th>Browser</th>
<th>Kernel</th>
<th>Nav.</th>
<th>Sub-windows</th>
<th>Frames</th>
<th>Display policy</th>
<th>Display mech.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>microkernel</td>
<td>isolated</td>
<td>isolated</td>
<td>not isolated</td>
<td>none</td>
<td>streaming image</td>
</tr>
<tr>
<td>Chrome</td>
<td>monolithic</td>
<td>isolated</td>
<td>different-site</td>
<td>not isolated</td>
<td>none</td>
<td>custom</td>
</tr>
<tr>
<td>Gazelle</td>
<td>monolithic</td>
<td>different-origin</td>
<td>isolated</td>
<td>different-origin</td>
<td>opaque overlay</td>
<td>streaming image</td>
</tr>
<tr>
<td></td>
<td>microkernel</td>
<td>isolated</td>
<td>isolate</td>
<td>different-origin</td>
<td>delegate once</td>
<td>window manager</td>
</tr>
</tbody>
</table>
