Malware Defense: New Trends and Approaches

Dawn Song

UC Berkeley
Malicious Code: Critical Threat

- Worms
- Viruses
- Trojan Horses
- Spyware
- Botnets
- Rootkits
Growth of New Malicious Code Threats

(source: Symantec)
Malicious Code: Critical Threat

- Worms
- Viruses
- Trojan Horses
- Spyware
- Botnets
- Rootkits
Outline

• Malware: Emerging Threats

• Defense: New Approaches
Malware enters new landscape as more parts of the world get connected
Changing Medical Device Landscape

- More medical devices are becoming networked
- Increased software complexity
  - Software plays an increasing role in device failure
    » 2005-2009 (18%) due to software failure, compared to (6%) in 1980s
- Medical device hardware and software is usually a *monoculture* within device model
The Population of AEDs Has Increased Significantly Over the Past 5 Years

28,000 adverse event reports in 14 Models recalled 2005-2010.

The case for Software Security Evaluations of Medical Devices [HealthSec’11]
Case Study

• Cardiac Science G3 Plus model 9390A

• Analysis
  – Manual reverse engineering using IDA Pro
    » MDLink, AEDUpdate and device firmware
  – Automatic binary analysis
    » BitBlaze binary analysis infrastructure
    » BitFuzz, the dynamic symbolic execution tool

• Vulnerabilities discovered
  1. AED Firmware - Replacement
  2. AEDUpdate - Buffer overflow
  3. AEDUpdate - Plain text user credentials
  4. MDLink - Weak password scheme

*The case for Software Security Evaluations of Medical Devices [HealthSec’11]*
Firmware Replacement

- Firmware update uses custom CRC to verify firmware

- Modified firmware, with proper CRC, is accepted by AED and update software

- Impact: Arbitrary firmware

DEVICE COMPROMISED

The case for Software Security Evaluations of Medical Devices [HealthSec’11]
AEDUpdate Buffer Overflow

- During update device handshake, device version number exchanged
- AEDUpdate *improperly* assumes valid input
- Enables *arbitrary* code execution
  - Data sent from AED can be executed as code on the host PC

*The case for Software Security Evaluations of Medical Devices [HealthSec’11]*
Initial Malicious Firmware Update

The case for Software Security Evaluations of Medical Devices [HealthSec’11]
Consumer-grade BCI Devices

- Price: ≈ 300 USD
**Exercise Equipment for Your Mind**
Experts agree that the human brain should be exercised like other body elements. Use the MindWave with specially designed neuroscience meditation, mental fitness and game applications on your home PC or Mac.

**Blink Challenge**
Uses an Emotiv interface and it can catch your blink immediately. Try to beat your longest stare! Or how fast can you blink? You just wear the headset and try this game.

**Arena**
A game that requires you to use the power of your mind against your opponent. To play the game, you must first train your mind to shoot fireballs using the Emotiv PUSH command.

**Spirit Mountain Demo Game**
Experience the fantasy of having supernatural powers and controlling the world with your mind. Your journey will take you through a mythical landscape of forests, temples and an environment that adjusts itself based on how you feel.

**Master Mind**
Master Mind allows users to play their favorite PC games with the power of their mind. Existing PC games such as World of Warcraft™ and Call of Duty™ can now be played with the power of your mind.

**Mind Mouse**
Mind Mouse is a revolutionary thought-controlled software application which allows the user to navigate the computer, click and double-click to open programs, compose email and send with the power of their mind.

**Emotiv EPOC Unity3D Developer Support Pack**
This package contains a full Unity3D™ Wrapper for the Emotiv EPOC EmoEngine API and a working demonstration game project and assets.

Rate this product:

<table>
<thead>
<tr>
<th><strong>HEADSET &amp; ACCESSORIES</strong></th>
<th><strong>DEVELOPER &amp; RESEARCH PACKAGES</strong></th>
<th><strong>APP STORE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Blink Challenge</strong></td>
<td><strong>Master Mind</strong></td>
<td><strong>Mind Mouse</strong></td>
</tr>
<tr>
<td>$4.95</td>
<td><strong>$99.00</strong></td>
<td><strong>$99.00</strong></td>
</tr>
<tr>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
</tr>
<tr>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
</tr>
<tr>
<td><strong>Arena</strong></td>
<td><strong>$14.95</strong></td>
<td></td>
</tr>
<tr>
<td><strong>BUY NOW</strong></td>
<td><strong>$79.95</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Blink Challenge</strong></td>
<td><strong>Master Mind</strong></td>
<td><strong>Mind Mouse</strong></td>
</tr>
<tr>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
</tr>
<tr>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
<td><strong>BUY NOW</strong></td>
</tr>
<tr>
<td><strong>Spirit Mountain Demo Game</strong></td>
<td><strong>$4.95</strong></td>
<td></td>
</tr>
<tr>
<td><strong>FREE</strong></td>
<td><strong>FREE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DOWNLOAD</strong></td>
<td><strong>BUY NOW</strong></td>
<td></td>
</tr>
</tbody>
</table>
What if an EEG gaming app is malicious?

Secretly reading your mind?
BCI as Side-Channel to the Brain

• Experiment objective:
  – Can the signal captured by a consumer-grade EEG device be used to extract potentially sensitive information from the user?

• Experiment setup:
  – 30 EECS students (28)
    » 18 male and 10 female
  – Minimal information: did not provide experiment objective
  – Experiments lasted about 45 minutes per participant
    » Each experiment lasted about 90 seconds
    • Flashing of multiple images on the screen

On the Feasibility of Side-Channel Attacks with Brain-Computer Interfaces [USENIX Security’12]
Experiment Methodology

On the Feasibility of Side-Channel Attacks with Brain-Computer Interfaces
[USENIX Security’12]
Attack Stimuli

Information tested:
• First digit of PIN
• Do you know this person?
• Do you have an account at this bank?
• What month were you born in?
• Where do you live?

On the Feasibility of Side-Channel Attacks with Brain-Computer Interfaces
[USENIX Security’12]
Experimental Results

On the Feasibility of Side-Channel Attacks with Brain-Computer Interfaces
[USENIX Security’12]
Outline

- Malware: Emerging Threats
- Defense: New Approaches
Defenses

Reactive Approaches

Detecting:
- Hidden code
- Privacy/sensitive data leakage
- Trigger-based behavior
- Hooking behavior

Detecting:
- Code reuse/repackage
- In-App Billing Vulnerability
- Permission misuse
- Security spec violation

BitBlaze Binary Analysis Infrastructure

DroidBlaze Analysis Infrastructure
Defenses

- Reactive Approaches
- Offensive Approaches
- Proactive Approaches
Finding Vulnerabilities in Malware

• Attackers exploit vulnerabilities in benign software

• Does malware have vulnerabilities?

• Can we find vulnerabilities in malware?

• New arsenal to combat malware
  – Cleaning hosts
  – Malware genealogy
  – Botnet infiltration & take-down
  – Cyber warfare
Finding Implementation Vulnerabilities in Malware

- Decomposition-&-restitching dynamic symbolic execution [BitBlaze]
- Compare Stitched vs. Vanilla explorations
  - Run both on same malware for 10 hours and find bugs

<table>
<thead>
<tr>
<th>Name</th>
<th>Vulnerability Type</th>
<th>Encoding function</th>
<th>Search Time (Stitched)</th>
<th>Search Time (Vanilla)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zbot</td>
<td>Null dereference</td>
<td>checksum</td>
<td>17.8 sec</td>
<td>&gt;600 min</td>
</tr>
<tr>
<td>Zbot</td>
<td>Infinite loop</td>
<td>checksum</td>
<td>129.2 sec</td>
<td>&gt;600 min</td>
</tr>
<tr>
<td>MegaD</td>
<td>Process Exit</td>
<td>decryption</td>
<td>8.5 sec</td>
<td>&gt;600 min</td>
</tr>
<tr>
<td>Gheg</td>
<td>Null dereference</td>
<td>weak decryption</td>
<td>16.6 sec</td>
<td>144.5 sec</td>
</tr>
<tr>
<td>Cutwail</td>
<td>Heap Corruption</td>
<td>none</td>
<td>39.4 sec</td>
<td>39.4 sec</td>
</tr>
</tbody>
</table>

*Input Generation via Decomposition and Re-Stitching: Finding Bugs in Malware [CCS’10]*
Experimental Results: Bug Persistency

• Each malware family comprises many binaries over time
  – Packing, functionality changes …
• Bugs have been present in malware families for long time

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Binaries</th>
<th>Bug reproducibility</th>
<th>Newest</th>
<th>Oldest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gheg</td>
<td>5</td>
<td>~9.5 months</td>
<td>Nov. 28, 2008</td>
<td>Feb. 6, 2008</td>
</tr>
<tr>
<td>Cutwail</td>
<td>2</td>
<td>~3 months</td>
<td>Nov. 5, 2009</td>
<td>Aug. 3, 2008</td>
</tr>
</tbody>
</table>

*Input Generation via Decomposition and Re-Stitching: Finding Bugs in Malware [CCS’10]*
Protocol Model Inference & Finding Vulnerabilities in Botnet C&C Protocols

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
Automatic Protocol Model Inference for MegaD

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
App 1: Disabling Botnets

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
App 1: Disabling Botnets

- Identify Critical Links

- Significance
  - Taking down 1 MegaD SMTP Server
  - Stops bots spam across multiple MegaD C&C server groups
  - Validated through experiment

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
App 2: Identify MegaD SMTP Servers

MegaD’s Fake SMTP Server

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
App 2: Implementation Differences

Fingerprint & Identify MegaD SMTPs in the wild

Postfix SMTP 2.5.5

MegaD SMTP

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
App 3: Identify Design Flaws

- Real bot goes through **long red path** to obtain spam templates
- Fake bot may use shortcut: 0 -> 1, bypassing Master Server to loot spam templates
- Application [Botnet Judo]: Update spam filtering rules *before* spam is sent out

Inference and Analysis of Formal Models of Botnet Command and Control Protocols [CCS’10]
Defenses

- Reactive Approaches
- Offensive Approaches
- Proactive Approaches
New Security Primitives

• For building secure systems even when the machine may be compromised
  – Cloud Terminal [USENIX Annual Technical Conf’12]

• For building secure applications by design
  – Context-sensitive auto-sanitization in web templating languages using type qualifiers [CCS’11]

• For better security architecture & auditability
  – Privilege separation in HTML5 [USENIX Security’12]
Goal: Trusted Path into the Cloud

• How to securely access & interact with cloud applications?
  – E.g., online banking, enterprise apps
• Quickly switch your PC to a secure operation mode
• Application provides a normal GUI
• But, information security does not depend on primary OS or its software
  – Even if commodity OS is compromised by malware
Cloud Terminal Architecture

Secure Thin Terminal (STT)
- Untrusted OS
- Untrusted helper
- Microvisor
- Hardware and TPM

Cloud Terminal client

Cloud Terminal protocol
- Mutual authentication
- Display and input
- Remote attestation
- Transport security

Cloud Rendering Engine (CRE)
- Application VM
  - VNC server
- Application VM
  - VNC server
- Dispatcher

Encrypted tunnel

Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems [USENIX ATC’12]
## Advantages over Existing Approaches

<table>
<thead>
<tr>
<th>Property</th>
<th>Red/Green VMs</th>
<th>Per-App VMs</th>
<th>Browser OS (Chrome)</th>
<th>VDI &amp; Thin Client</th>
<th>Flicker</th>
<th>Cloud Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installable w/existing OS</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Attestation</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Generic Apps</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Fine-grained isolation</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>No trust in host OS</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>User interface</td>
<td>any</td>
<td>any</td>
<td>browser</td>
<td>any</td>
<td>✗</td>
<td>any</td>
</tr>
<tr>
<td>Mgmt. effort</td>
<td>med.</td>
<td>high</td>
<td>low</td>
<td>low</td>
<td>low</td>
<td>low</td>
</tr>
<tr>
<td>TCB size (LOC)</td>
<td>&gt;1M</td>
<td>&gt;1M</td>
<td>&gt;1M</td>
<td>&gt;1M</td>
<td>250 + app logic</td>
<td>22K</td>
</tr>
</tbody>
</table>

*Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems [USENIX ATC’12]*
Evaluation: client TCB

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microvisor</td>
<td>7.7K</td>
</tr>
<tr>
<td>Terminal client</td>
<td>3.0K</td>
</tr>
<tr>
<td>Crypto (PolarSSL)</td>
<td>5.5K</td>
</tr>
<tr>
<td>Attestation (Flicker)</td>
<td>5.7K</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>21.9K</strong></td>
</tr>
</tbody>
</table>

*Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems [USENIX ATC’12]*
Evaluation: performance

- 16 core, 64GB server, 670 mi from client
- Simultaneous clients replay recorded usage

<table>
<thead>
<tr>
<th>App</th>
<th>Activity</th>
<th>Baseline (ms)</th>
<th>Latency (ms) with # of clients =</th>
<th>Network usage (bytes)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>150</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>Edit</td>
<td>Launch</td>
<td>2,844</td>
<td>2,208</td>
<td>2,441</td>
<td>2,553</td>
</tr>
<tr>
<td></td>
<td>Type a key</td>
<td>30</td>
<td>53</td>
<td>50</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>Move mouse</td>
<td>32</td>
<td>49</td>
<td>59</td>
<td>51</td>
</tr>
<tr>
<td>PDF</td>
<td>Launch</td>
<td>1,699</td>
<td>2,093</td>
<td>2,147</td>
<td>2,493</td>
</tr>
<tr>
<td></td>
<td>Scroll</td>
<td>114</td>
<td>1,270</td>
<td>1,380</td>
<td>1,704</td>
</tr>
<tr>
<td>Bank</td>
<td>Launch</td>
<td>6,911</td>
<td>2,319</td>
<td>2,563</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>New page</td>
<td>1,183</td>
<td>2,610</td>
<td>2,661</td>
<td>---</td>
</tr>
<tr>
<td>Gmail</td>
<td>Launch</td>
<td>6,936</td>
<td>2,254</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>Display msg.</td>
<td>992</td>
<td>2,254</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems [USENIX ATC’12]
New Security Primitives

- For building secure systems even when the machine may be compromised
  - *Cloud Terminal [USENIX Annual Technical Conf’12]*

- For building secure applications by design
  - *Context-sensitive auto-sanitization in web templating languages using type qualifiers [CCS’11]*

- For better security architecture & auditability
  - *Privilege separation in HTML5 [USENIX Security’12]*
Web Vulnerabilities: A Growing Threat

Source: Database 2012
Can never find & fix all XSS vulnerabilities 😞

How to build web apps free of XSS vulnerabilities?
An Attack Example (XSS)

An attacker sends an email with a link to a malicious website.

```javascript
n.innerHTML = x;
x = "<img src='alice.gif'/>";
n.innerHTML = x;
```

The victim visits the malicious website and receives an image with the attacker's profile.
An Attack Example (XSS)

\[ \text{http://twitter.com#!'} \_\text{onerror=bad()} \]

\[ ' \_\text{onerror=bad()..} \]

\[ \text{<img src='} \_\text{onerror=bad()..} \_\text{.gif '} /> \]

\[ x = "\text{<img src='} + q + "\_\text{.gif'} />"; \]

\[ \text{n.innerHTML} \_\text{XSS} \]
Key Property: Structure Integrity

```
<img src=' ' onerror=bad()… '></img>
```

Intended Structure

```
<HTML>
  <BODY>
    <DIV>
      <IMG src=' ' onerror=bad()>
    </DIV>
  </BODY>
</HTML>
```

Actual Structure

```
<HTML>
  <BODY>
    <DIV>
      <IMG src=' ' onerror=bad()=
    </DIV>
  </BODY>
</HTML>
```
Structure Integrity Attacks

Web Languages Structure Integrity Attacks

- SQL
- JS & HTML
- HTTP URLs
- CSS
- SVG
- MIME Types

- SQL Injection
- XSS
- HTTP Parameter Pollution
- CSS-based XSS
- SVG-based XSS
- Content-Sniffing
Solution: Templates & Holes

```html
<img src='... onerror=bad()'... />
```
Today's Predominant Enforcement: Sanitization

Example

```php
print("<img src='\'>";
print(Sanitize(userimg));
print("\'></img>";
```

URL Encode

```
%E2%80%99%20onerror=%3Dalert(%E2%80%9CXSS%E2%80%9D);... %3B%E2%80%A6%0A
```
Challenges: Getting Sanitization Right

```
print("<img src='');
print(Sanitize(userimg));
print("/>");
```
Incorrect Sanitizer Choice

Attacks Vary By Parsing Contexts!
Incorrect Sanitizer Choice

Does manual sanitization really fail?

- Microsoft shipping .NET applications
  - 400,000 LOC
  - [Saxena et al. CCS’11]

Context-Mismatch Sanitization
Our Solution

```c
template ImgRender($imgLink, $name)
{................}
```

Context-Sensitive Auto-Sanitization

- URL Encode
- Param Encode
- Html Encode

How To Auto-Sanitize Existing Code?

- Compatible
- Auditable
- Secure
- Fast

Context-sensitive auto-sanitization in web templating languages using type qualifiers [CCS’11]
Key Ideas: Context Type Qualifier

- **Context Type Qualifier:**
  - "Which contexts is a string safe to be rendered in"

**Type Inference To Decide Sanitizer Placement**

```plaintext
x := "<img src='" . $imgLink;

y := UrlEncode($imgLink)

x := "<img src='" . y;
```

*Context-sensitive auto-sanitization in web templating languages using type qualifiers [CCS'11]*
Implementation

- Implemented in Google Closure Templates
- Handles Flow-sensitivity
- Much faster than Runtime Parsing
Adoption

# of Auto-sanitized Templates in Google production code

In Other Frameworks…

jQuery

写作更少，做得更多

GO

http://golang.org

GWT
New Security Primitives

• For building secure systems even when the machine may be compromised
  – Cloud Terminal [USENIX Annual Technical Conf’12]

• For building secure applications by design
  – Context-sensitive auto-sanitization in web templating languages using type qualifiers [CCS’11]

• For better security architecture & auditability
  – Privilege separation in HTML5 [USENIX Security’12]
One security principal with ambient authority (privileges)
(Secure Computing Research for Users' Benefit)

Institute and Technology Center for Secure Computing

The Institute and Technology Center for Secure Computing focuses on scientific research to make computer systems and devices safe and secure for users.

The University of California, Berkeley, with participation from world-leading researchers from Drexel University, Duke University, University of Illinois at Urbana-Champaign, and Intel.

Innovative software and hardware architectures can provide better security and make it harder for malware, by building on a trusted software layer that manages security for the entire computing system. These architectures can provide security for mobile computing, especially focusing on smartphones and tablets.

Third-party apps are safe for users. Innovative system architectures that can protect personal data in complex distributed systems and cloud environments.

We are looking into ways to give people more control over their personal data and wherever it may be stored.

We offer security analytics to manage and measure a site's security and to enable systems to adapt to threats.

If you're interested in our agenda, you are invited to check out our videos or read our white paper.
SCRUB (Secure Computing Research for Users' Bene
The Intel Science and Technology Center for Secure Computing

Home

Home

The Intel Science and Technology Center for Secure Computing focuses on scientific research to make computing technology safe and secure for users.

We are headquartered at the University of California, Berkeley, with participation from world-leading researchers from Carnegie Mellon University, Drexel University, Duke University, University of Illinois at Urbana-Champaign, and Intel. The center is funded by Intel, and includes both academics and Intel researchers working together collaboratively to make computing safer for users.

The center is actively engaged in several research directions:

- We are studying how innovative software and hardware architectures can provide better security and make personal computers safer from malware by building on a trusted software layer that manages security for the entire platform.
- We are studying how to provide security for mobile computing, especially focusing on smartphones and tablets. Our goal is ensuring that third-party apps are safe for users.
- We are studying novel system architectures that can protect personal data in complex distributed systems and help avoid data breaches. We are looking into ways to give people more control over their personal data and make it more secure, wherever it may be stored.
- We are developing security analytics to manage and measure a site’s security and to enable systems to adapt to new threats.

To learn more about the center’s agenda, you are invited to check out our videos or read our white paper.
580KB of code
permissions ?
all data on all websites
580KB of code TCB (javascript)
The Problem

• #1: bundling
  – one origin, two applications
Screenshot Component can save files (doesn’t need to)
Image Editor can take screenshots (doesn’t need to)
Not the exception

19 out of top 20 extensions exhibited this behavior
The Problem

• **#1: Bundling**
  – One origin, two applications

• **#2: TCB inflation**
  – All code runs with full privileges
  – Only core application needs to
580KB of TCB

500KB generic libraries (jquery, jquery-ui, ...
Not the exception

We measured the fraction of functions requiring privileges
For ~50% of extensions
< 5% of functions require privileges

Data collected from the Top 50 Chrome Extensions
For ~80% of extensions, < 20% of functions require privileges.

Data collected from the Top 50 Chrome Extensions.
Our Solution: privilege separation
Only the parent runs privileged calls based on a policy. For example, Image Editor not allowed to capture screenshots.
User clicked menu button -> Privileged Parent

Screenshot Component

User clicked menu button

call captureVisibleTab

TypeError: Cannot read property 'captureVisibleTab' of undefined
chrome.tabs.captureVisibleTab
sendToParent('captureVisibleTab')
Message Listener
chrome.tab.captureVisibleTab
sendToChild(returnValue)
Policy Code
Message Listener
Application gets return value
Screenshot Component
Privileged Parent
Seamless Proxying
Code Shim
Image Editor Component

chrome.tabs.
captureVisibleTab

sendToParent
(‘captureVisibleTab’)

Message Listener

Application gets ‘denied’

Policy Code

Message Listener

sendToChild( ‘denied’)

Privileged Parent
parent invariants
the parent can’t convert string to code
the parent can’t execute arbitrary code from the web
the parent is the only entry point into the privileged origin
only primitive data types cross the privilege boundary
<table>
<thead>
<tr>
<th>Application</th>
<th>Number of Users</th>
<th>Initial TCB (KB)</th>
<th>New TCB (KB)</th>
<th>Lines Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Awesome Screenshot</td>
<td>802,526</td>
<td>580</td>
<td>16.4</td>
<td>0</td>
</tr>
<tr>
<td>SourceKit</td>
<td>14,344</td>
<td>15,000</td>
<td>5.38</td>
<td>13</td>
</tr>
<tr>
<td>SQL Buddy</td>
<td>45,419</td>
<td>100</td>
<td>2.67</td>
<td>11</td>
</tr>
</tbody>
</table>
Privilege separation in HTML5 applications shows how applications can cheaply create arbitrary number of components.

Our approach utilizes standardized abstractions already implemented in modern browsers.

We retrofit applications to demonstrate TCB reductions.
New Security Primitives

• For building secure systems even when the machine may be compromised
  – Cloud Terminal [USENIX Annual Technical Conf’12]

• For building secure applications by design
  – Context-sensitive auto-sanitization in web templating languages using type qualifiers [CCS’11]

• For better security architecture & auditability
  – Privilege separation in HTML5 [USENIX Security’12]
Conclusion

Malware enters new landscape as more parts of the world get connected

- Reactive Approaches
- Offensive Approaches
- Proactive Approaches
http://bitblaze.cs.berkeley.edu
http://webblaze.cs.berkeley.edu
dawnsong@cs.berkeley.edu