The Web/Local Boundary Is Fuzzy

A Security Study of Chrome’s Process-based Sandboxing

Yaoqi Jia, Zheng Leong Chua, Hong Hu, Shuo Chen, Prateek Saxena, Zhenkai Liang

National University of Singapore
Microsoft Research
Monolithic Browser Design

Web Page

Local System

Files

Apps

Sensors

EXE
2\textsuperscript{nd} Generation Browser: Process-based Isolation

- Process-based sandboxing – process boundary

Web Page

Web/Local Boundary

Browser \quad \text{kernel}

Local System
Is the Web/Local Boundary Sufficient?

- Used by most modern browsers

Web Page A

Web Page B

Web/Local Boundary

Browser kernel

Local System

Stress testing, bug bounty, fuzzing ...
Contributions

• The Web/Local Boundary is Fuzzy!

Concrete Attacks
• Access local files, system control
• Use 1 bug in renderer process

Attack Details
• Bypass in-memory protections using data-oriented attacks

Solutions
• Imperfect existing solutions
• Our light-weight mitigation
The Web/Local Boundary is Fuzzy

• Landscape changes --- Rise of the cloud services

Before → 2016

Web Page

Fuzzy

Cloud

A new path to access local system

Web/Local Boundary

Browser Kernel

Local System

Client
Attacks due to Fuzzy Web/Local Boundary
Attack Example 1: Drop a Malware

www.evil.com

Browser Kernel

Local System

Dropbox Cloud

Dropbox Client
Example 2: Steal a Local File

www.evil.com

Browser Kernel

Local System

Dropbox

Cloud

Dropbox

Client
Example 3: Install Malware

www.evil.com

Browser Kernel

Local System (Android)

Google Play Server

Google Play
Example 4: Remote System Control

www.evil.com

Browser Kernel

Local System (VM)

OpenStack

OpenStack Server

Vnc

rm -rf /

> rm -rf /
But ... Chrome’s Protections

- Same-Origin Policy (SOP)
- Control-Flow Integrity (CFI) *on the way*
- In-Memory Partitioning
- Internal Randomization
SOP Enforcement in Chrome

www.evil.com

bool SecurityOrigin::canAccess() {
    if (m_universalAccess)
        return true;
    if (this == other)
        return true;
    ....
    return canAccess;
}

Various SOP checks for cross-origin read/write: contentDocument, frames, etc.
Control-Flow Integrity

- CFI: control flows cannot be modified (on the way)

```c
<func1>
    ...
    lea func2, %eax
    ...
    jmp *%eax
    ...
<func2>
    push %ebx
    ...
```
Bypass SOP & CFI

- Corrupt critical data
- Not modify control flow
- Bypass SOP checks

```cpp
bool SecurityOrigin::canAccess() {
    if (m_universalAccess)
        return true;
    if (this == other)
        return true;
    ....
    return canAccess;
}
```

When `m_universalAccess` is true, the check always passes
In-Memory Partitioning

- Separate different types of objects in 4 partitions
- Surrounded by inaccessible guard pages
Cross-Partition References to Bypass Partitioning

- Link objects in one partition to another
- Pervasive & often under the control of scripts
  - Dereference pointers to cross partition boundaries
Partition-based Randomization

- Randomize the base address of each partition
- Guard pages cannot be read/written
Fingerprinting Technique to Bypass ASLR & Find Critical Data

- Special pattern for security monitor objects
- Linearly scan memory

```cpp
class PLATFORM_EXPORT SecurityOrigin {
    ......
    String m_protocol;
    String m_host;
    String m_domain;
    String m_suboriginName;
    unsigned short m_port;
    bool m_isUnique;
    bool m_universalAccess;
    bool m_domainWasSetInDOM;
    bool m_canLoadLocalResources;
    bool m_blockLocalAccessFromLocalOrigin;
    bool m_needsDatabaseIdentifierQuirkForFiles;
};
```

Match the pattern

```
......
B9 BB 88 20
B9 CC 91 10
B9 CC 91 10
00 00 00 00
00 00 00 00
......
```

- protocol
- host
- domain
- suborigin
- m_universalAccess
Find the Address of Vulnerable Array

- Create a predictable “fingerprinting” object
- Linearly scan memory to find the object’s location

\[ \text{Addr}_{\text{base}} = \text{Addr}_{\text{obj}} - \text{Offset} \]
Bypass SOP & In-Memory Protections

• SOP
  Data-oriented attacks

• CFI
  Data-oriented attacks

• In-memory partitioning
  Cross-partition references

• Internal ASLR
  Fingerprinting technique

Seems difficult to bypass
Attack Implementation

- Work on proper memory error vulnerabilities
  - POC: CVE 2014-1705 heap overflow in V8 (Chrome 33)
- Over 10 SOP-related flags (Chrome 45)
- End-to-end attacks
  - Access files on the local system
    - Dropbox, Google Drive
  - Interact with local system
    - OpenStack, Google Play
  - Misuse system sensors
    - Fitbit, Runkeeper
Protections against Web/Local Attacks
Web Browser-Side Protection

• Memory safety
  ✓ Huge code base, e.g., +5 million LOC for Chrome

• Software-based fault isolation (SFI)
  ✓ Cross-partition references
Light-Weight Mitigation

• Identify critical data
• ASLR to hide the address of critical data
  ✓ Address of the critical data is not saved in user space
  ✓ Average 3.8% overhead
• Raise the bar of Web/Local attacks
Disclosure to Google

• Fine-grained process-based isolation
  ➢ Chrome’s Out-of-Process iframes
  ➢ Performance overhead and massive refactoring

www.evil.com

www.dropbox.com

www.evil.com

www.dropbox.com
Cloud Service-Side Protection

- Distinguish requests of its site from client
- Restrict the privileges for the web interface
- Require the user’s consent

Diagram:
- Web Page
- Cloud
- Browser Kernel
- Local System
- Client
- Web/Local Boundary
Conclusion

• Concrete Attacks on Web/Local Boundary
  ✓ Access local files, system control
  ✓ Using 1 bug in renderer process

• Attack Details
  ✓ Bypass in-memory protections

Video at https://youtu.be/fIHaIQ4btok

POC at https://github.com/jiayaqijia/Web-Local-Attacks

• Solutions
  ✓ Imperfect existing solutions
  ✓ Open to researchers
Thanks

Yaoqi Jia
(Graduating in 2017)
jiayaoqi@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~jiayaoqi/