



Stony Brook University

# **CSE 361: Web Security**

CSRF, XSS, SRI, and Sandboxing

Nick Nikiforakis



CSRF (Sea Surf)

# Regular Web site usage

*Behind the scenes*

← → × https://acmebank.com



Destination account:

Amount:

```
<form method="POST"
target=https://acmebank.com/transfer>
  <input type="text" name="acct-to">
  <input type="text" name="amount">
  <input type="submit">
</form>
```



# Forcing browser to perform an action for the attacker



A screenshot of a web browser window. The address bar shows the URL `http://kittenpics.org`. The main content area displays a photograph of a small, fluffy kitten. Below the image, a code block contains the following HTML and JavaScript code:

```
<form method="POST" action="https://acmebank.com/transfer" id="transfer">
  <input type="hidden" name="act-to" value="987-654-3210">
  <input type="hidden" name="amount" value="100000">
</form>
<script>
transfer.submit()
</script>
```



Processing transaction

# Cross-Site Request Forgery (CSRF / "Sea Surf")

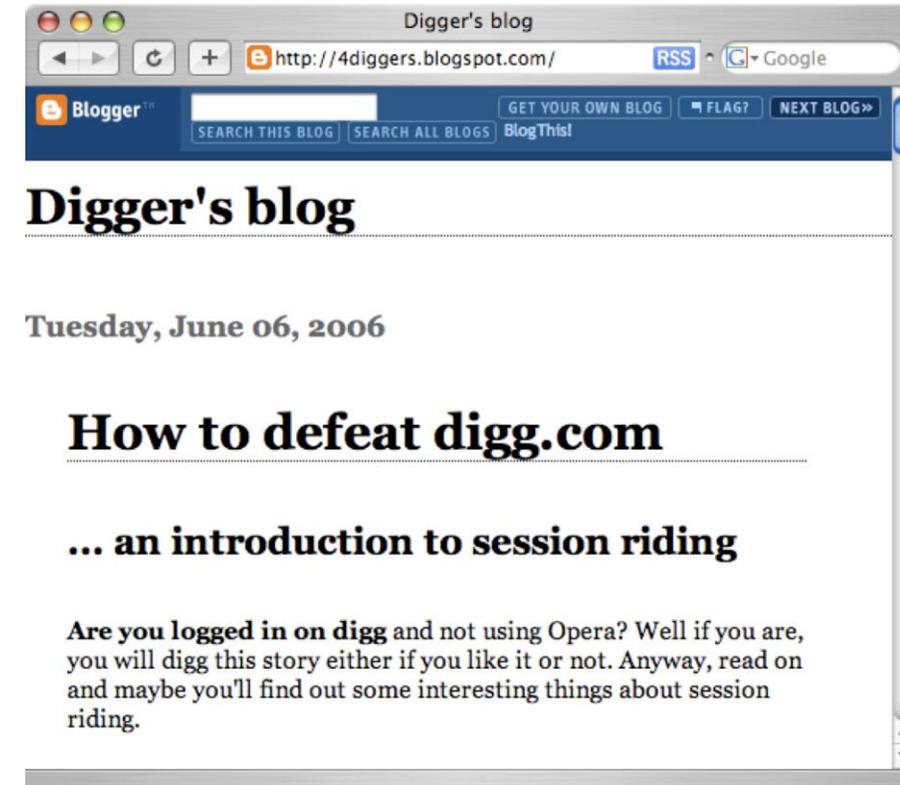
- Web application does not ensure that state-changing request came from "within" the application itself
- Attack works for GET ...
  - Image tag with src attribute:

```

```
  - Hidden iframes, css files, scripts, ...
- and POST
  - create iframe (or pop-up window)
  - fill created viewport with prefilled form
  - submit form

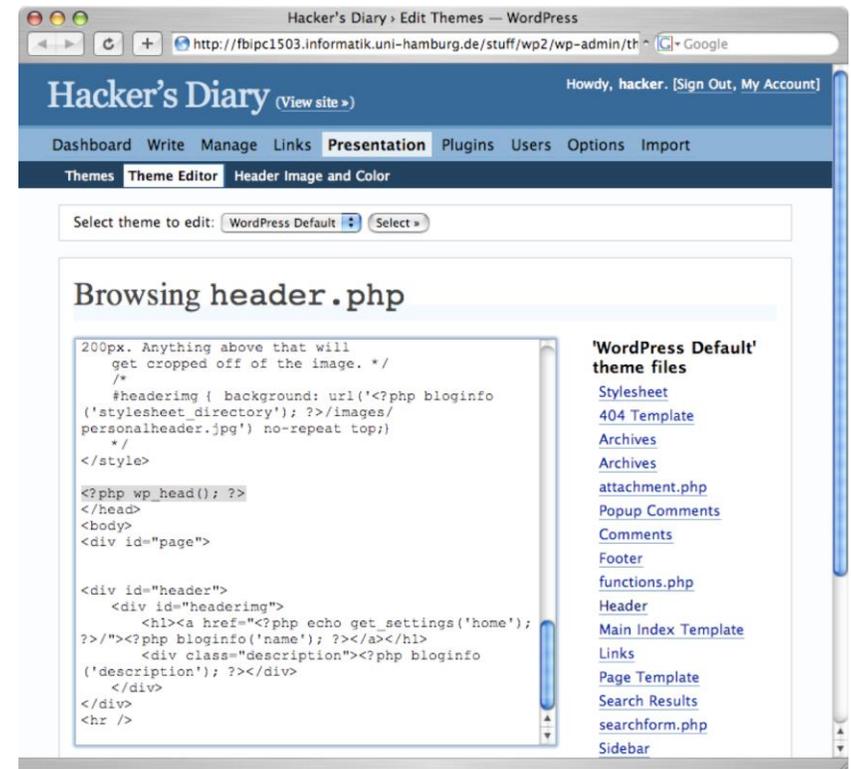
# CSRF Examples: digg.com (2006)

- digg.com determines frontpage based on how many "diggs" a story gets
- vulnerable against CSRF, could be used to digg an URL of the attacker's choosing
- Guess which article made it to the front page...



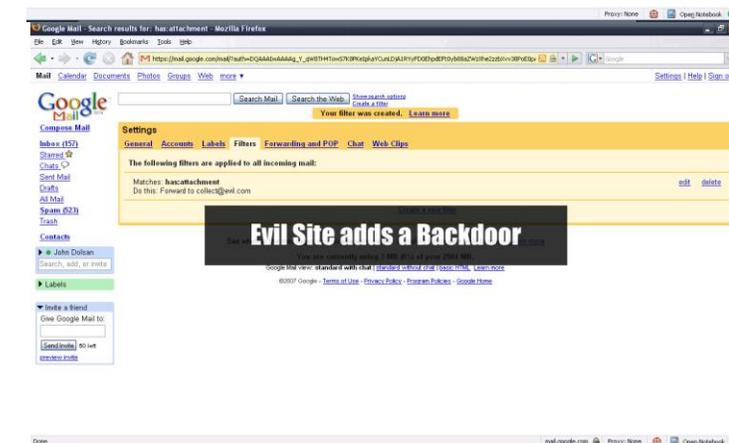
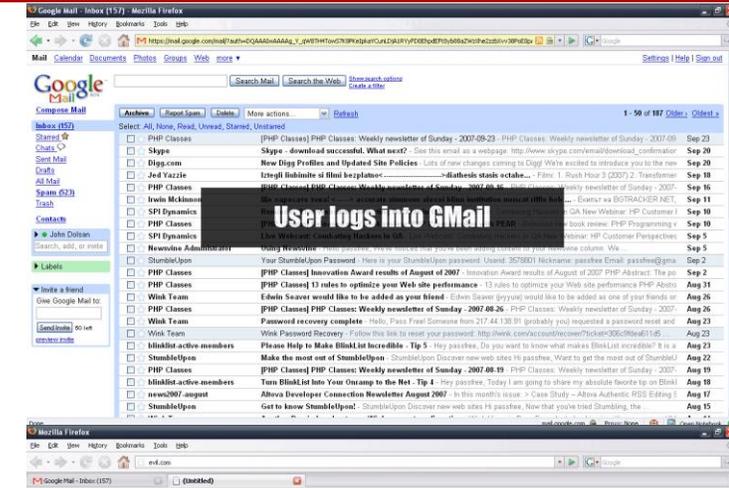
# CSRF Example: WordPress < 2.06 (2007)

- WordPress theme editor was susceptible
- WordPress themes are PHP files
- Attacker could modify files when logged-in admin visited his page
  - arbitrary code execution on targeted page



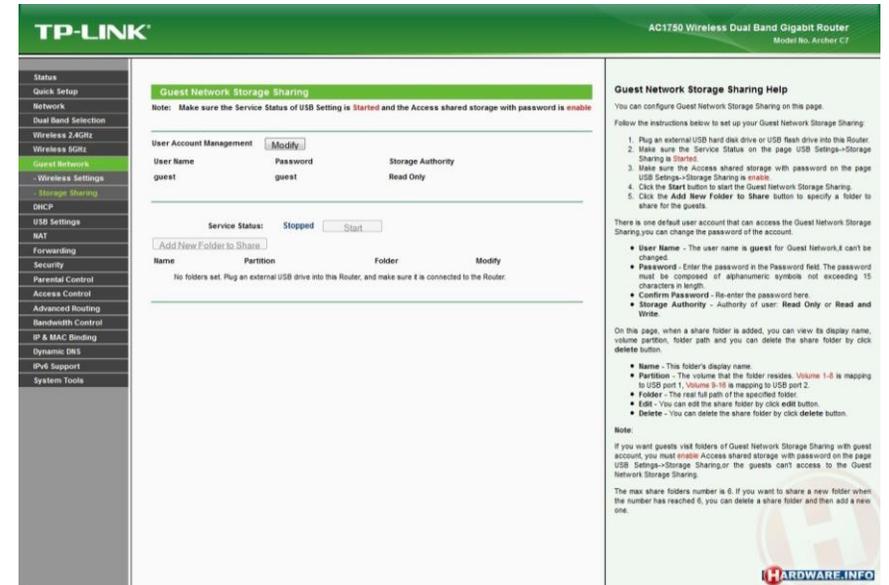
# CSRF Example: Gmail filters (2007)

- Google Mail insufficiently protected against CSRF
- Attacker could add mail filters
  - e.g., forward all emails to a certain address
- According to a victim, this led to a domain takeover
  - Attacker adds redirect filter
  - Attacker request AUTH code for domain transfer



# CSRF Example: TP-Link routers (CVE-2013-2645)

- TP-Link Web interface was vulnerable to configuration changes via CSRF
  - set root of built-in FTP server, enable FTP via WAN, ...
  - modify DNS server
- Exploited in the wild to change DNS server
  - redirects all DNS traffic to attacker's server
  - leaking all visited domains
  - allowing for trivial MitM attacks
- Only worked when user was logged in



# CSRF in 2017 to 2019

- CVE-2017-7404 D-Link router, firmware upload possible
- CVE-2017-9934 Joomla! CSRF to XSS
- CVE-2018-100053 LimeSurvey Delete Themes
- CVE-2018-6288 Kaspersky Secure Mail Gateway Admin Account Takeover
- CVE-2019-10673 WordPress CSRF to change admin email, password recovery for full compromise



# (Not really) Preventing CSRF: Refer(r)er Checking

- CSRF entails cross-domain requests
  - in theory, these should carry a referrer
  - server could decide based on header
- In practice, there are several problems
  - Middleboxes/proxies might strip (complete URL is sent, privacy concerns)
  - Attacker may strip Referer header by
    - using a data: URL
    - Referrer-Policy header
- Utility vs. Security trade-off
  - what do we do when the header is not present?

# Preventing CSRF: Origin Header Checking

- Privacy-friendly version of Referer
  - Contains only the origin, not the complete URL
- Always sent along XMLHttpRequests and WebSockets
  - requires changing program logic to use these requests for state-changing operations
- In **modern browsers**, also sent along with any **cross-origin POST requests**
  - server should not necessarily rely on only having modern clients, though

*What the third-party website receives*

Mechanism	Sent URL
Referer	https://www.news.com/bl ahblah?foo=bar
Origin	https://www.news.com

# Regular Web site usage

*Behind the scenes*

← → X https://acmebank.com



Destination account:

Amount:

```
<form method="POST"
target=https://acmebank.com/transfer>
  <input type="text" name="acct-to">
  <input type="text" name="amount">
  <input type="hidden" name="tk" value="n73gn9ia345ntu">
  <input type="submit">
</form>
```



# Preventing CSRF: Using CSRF tokens/nonces

← → X http://kittenpics.org



```
<form method="POST" action="https://acmebank.com/transfer"
id="transfer">
  <input type="hidden" name="act-to" value="987-654-3210">
  <input type="hidden" name="amount" value="100000">
  <input type="hidden" name="tk" value="no clue">
</form>
<script>
transfer.submit()
</script>
```



Invalid token  
no transaction

# Preventing CSRF: Using CSRF tokens/nonces

- Server generates token randomly for user
  - stores currently valid token in session for user
- Tokens are placed in all forms
  - inaccessible to the attacker without an XSS due to the SOP
- On submission, checks server-side token against submitted token
  - only allows action if tokens match
- Assures that a request's origin must be in the same origin

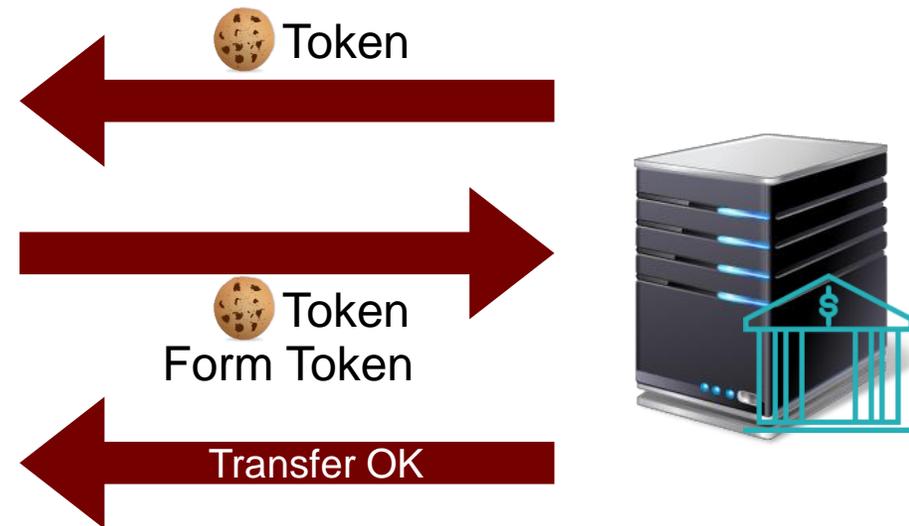
# Preventing CSRF: Double Submit Cookie

← → × https://acmebank.com



Destination account:

Amount:



# Preventing CSRF: Double Submit Cookie

- Require value in posted content to match value of certain cookie
  - generate token randomly on server, store in cookie
  - insert cookie's value into each form
    - server-side addition for protected forms or
    - via JavaScript after form was loaded
- Advantage: no server-side state required
  - just compare submitted form value against cookie
- Disadvantage: cookie tossing
  - If an attacker controls a subdomain, he might set token value
  - if the server only compares cookie and form token, CSRF protection is bypassed

# Preventing CSRF: Custom Headers

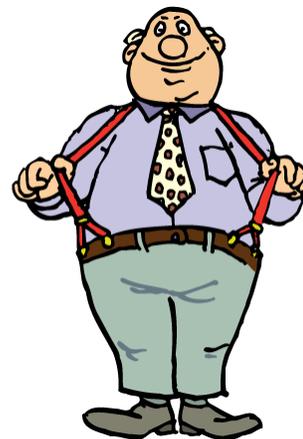
- Idea: use XMLHttpRequests for all state-changing requests
  - and attach a custom header (e.g., "X-CSRF-Free")
  - only handle requests with that header on the server
- Protection by existing technologies
  - Same-domain requests are always allowed
  - Cross-domain requests with custom headers requires pre-flight CORS request
- Advantage: no server-side state or randomness required
- Disadvantage: applications must be changed

# Preventing CSRF: Same-Site Cookies

- Two modes
  - **Strict**: even in top-level navigation, never send cookies with cross-origin request
    - if [facebook.com](https://www.facebook.com) set that, every user following a link there would not be logged in
  - **Lax**: non top-level navigation will not send cookies
    - cookies only send along with safe requests (GET, HEAD, OPTIONS, TRACE)
    - protects against POST-based CSRF, not against GET-based though
- Until May 2018 only supported by Chrome and Opera
- Since Chrome 80, defaults to SameSite=lax
  - SameSite=none only works with Secure flag

# CSRF Conclusion

- CSRF caused by servers accepting requests from outside their origin
  - hard to determine based on Referer header though
- CSRF can have severe effects
  - compromised firmware, hijacked Web sites, ...
- Several options for fixing exist
  - CSRF tokens nowadays implemented in any (good) framework
  - protection can be achieved using well-established principles (SOP, CORS)
  - SameSite cookies also address the issue, already default in Chrome
- Support still varies (<https://caniuse.com/?search=samesite>)
  - Use defense in depth

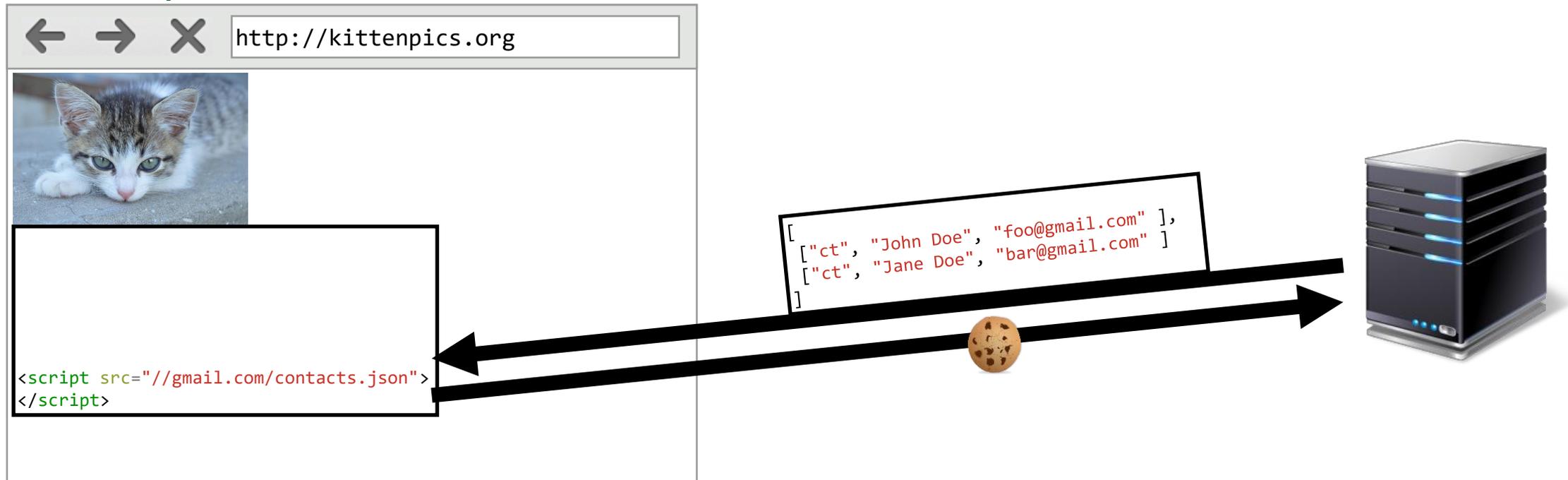


# Cross-Origin Data Leakage



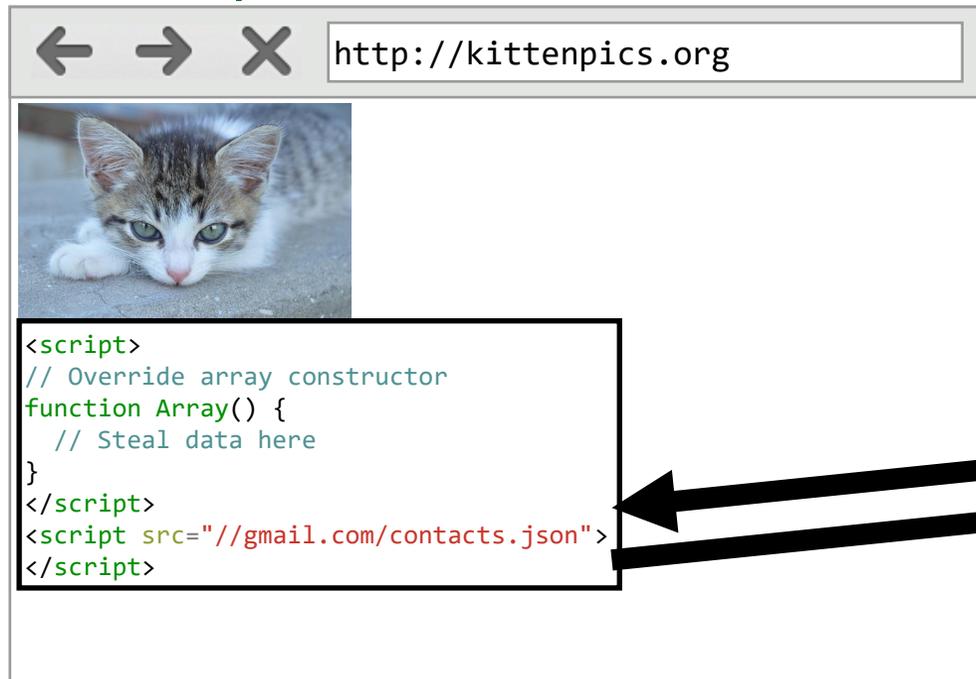
# JSON/JavaScript Hijacking (2006)

- Recall from previous lectures
  - script inclusion is exempt from SOP
  - all requests are made with cookies attached



# JSON/JavaScript Hijacking (2006)

- Recall from previous lectures
  - script inclusion is exempt from SOP
  - all requests are made with cookies attached



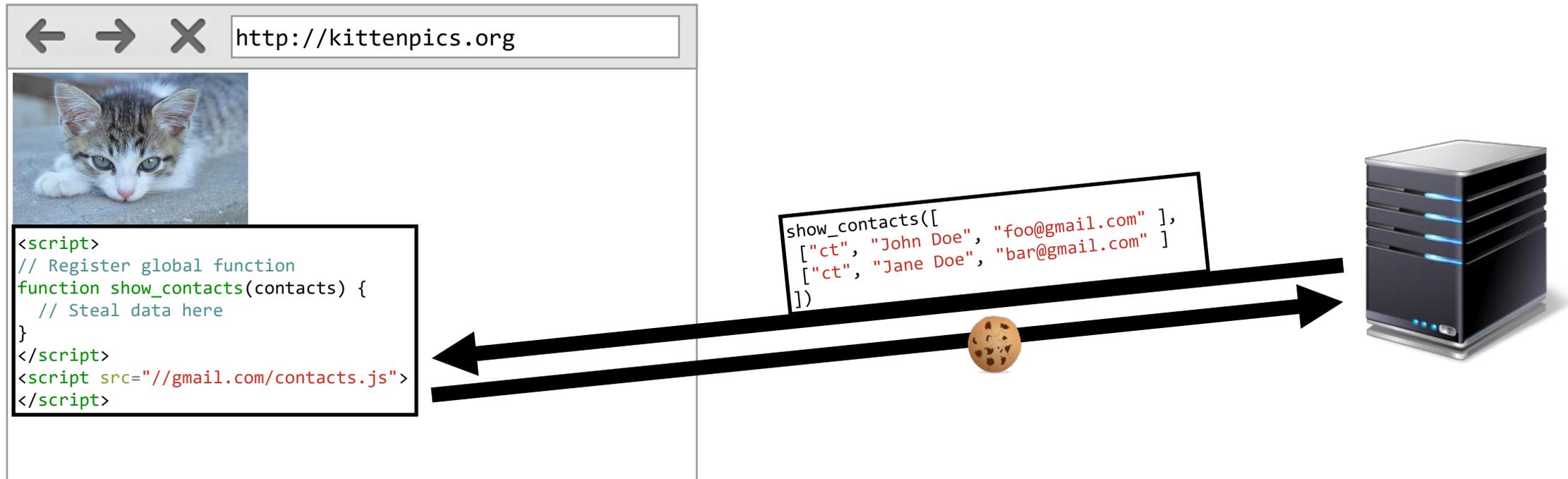
Based on browser quirks,  
fixed nowadays

```
[
  ["ct", "John Doe", "foo@gmail.com" ],
  ["ct", "Jane Doe", "bar@gmail.com" ]
]
```



# Cross-Site Scripting Inclusion (XSSI)

- Regular scripts may also be dynamically generated
  - We cannot read the source code, but can observe side-effects



# Exploiting XSSI

```
// Local variable at top level
var first_name = "John";
// Global variable due to missing var keyword
last_name = "Doe";
// Explicitly defined global variable
window.user_email = "john@doe.com";
```



---

```
console.log(first_name);
console.log(last_name);
console.log(user_email);
```

```
function example() {
  var email = "john@doe.com";
  window.MyLibrary.doSomething(email);
}
example();
```



---

```
window.MyLibrary = {};
window.MyLibrary.doSomething =
function(email) { console.log(email); }
```

# Exploiting XSSI

```
function example2() {  
  var secret_values = ["secret", "more secret"];  
  
  secret_values.forEach(function(secret) {  
    // do something secret in here  
  });  
}  
example2();
```



---

```
Array.prototype.forEach = function(callback) {  
  // "this" is bound secret_values  
  console.log(this);  
}
```

```
(function() {  
  function test(someInput) {  
    var email = "john@doe.com";  
    doNothingWithEmail(someInput);  
  }  
  
  test.call(someThing, "myInput");  
})();
```



---

```
Function.prototype.call = function() {  
  // "this" is bound test  
  console.log(this.toString());  
};
```

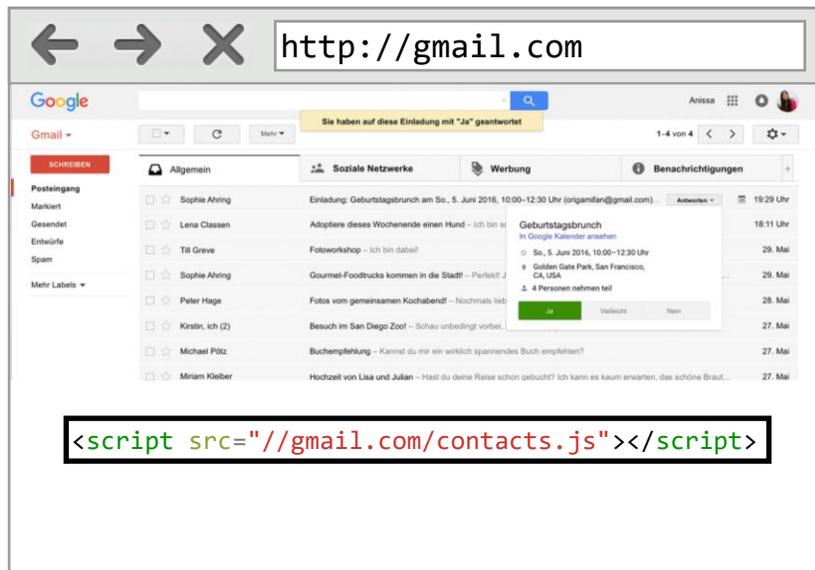
# Exploiting XSSI

- Trivial case: global variables registered
  - simply access the variable (registered in global scope of site)
- Little more involved: global function called
  - overwrite function (if necessary, create object before)
- Local variables accessible if functions are called on them
  - overwrite prototype
  - e.g., forEach or call



# Identifying potential XSSI [USENIX15]

- On each page visit, request included scripts twice
  - with and without cookies
- Diff the two results



```

545     .AddComment i18n("Tip: use
546     .Comment.Visible = False
547     End If
548     End With
549     With Cells(i + 3 + 3, 2)
550     .Name = ans(i)
551     .Interior.ColorIndex = COLOUR
552     .BorderAround xlDash, xlThis,
553     .Locked = False
554     If multiline(i) Then
555     .WrapText = True
556     .VerticalAlignment = xlTop
557     .EntireRow.RowHeight = 24
558     End If
559     End With
560     End With
561     Next i
562     ActiveWindow.DisplayHeadings = False
563     Range("PROJECT_NAME").Select
564     End With
565     MyProtect wks
566     Next i
567     ActiveWindow.DisplayHeadings = False
568     ActiveWindow.Zoom = 80
569     End With
570     End With
571     .Locked = False
572     If multiline(i) Then
573     .AddComment i18n("Tip: use
574     .IndestLevel = 1
575     .Comment.Visible = False
576     .WrapText = True
577     .VerticalAlignment = xlTop
578     .EntireRow.RowHeight = 80
579     Else
580     .VerticalAlignment = xlV
581     .HorizontalAlignment = xl
582     .EntireRow.RowHeight = 72
583     End If
584     End With
585     With Cells(i + 2 + 3, DATACOL - 1)
586     .Value = ""
587     .VerticalAlignment = xlTop
588     End With
589     Next i
590     ActiveWindow.DisplayHeadings = False
591     ActiveWindow.Zoom = 80
592     End With
593     End With
594     Next i
595     ActiveWindow.DisplayHeadings = False
596     ActiveWindow.Zoom = 80
597     End With
598     End With
599     End With
600     End With
601     End With
602     End With
603     End With
604     End With
605     End With
606     End With
607     End With
608     End With
609     End With
610     End With
611     End With
612     End With
613     End With
614     End With
615     End With
616     End With
617     End With
618     End With
619     End With
620     End With
621     End With
622     End With
623     End With
624     End With
625     End With
626     End With
627     End With
628     End With
629     End With
630     End With
631     End With
632     End With
633     End With
634     End With
635     End With
636     End With
637     End With
638     End With
639     End With
640     End With
641     End With
642     End With
643     End With
644     End With
645     End With
646     End With
647     End With
648     End With
649     End With
650     End With
651     End With
652     End With
653     End With
654     End With
655     End With
656     End With
657     End With
658     End With
659     End With
660     End With
661     End With
662     End With
663     End With
664     End With
665     End With
666     End With
667     End With
668     End With
669     End With
670     End With
671     End With
672     End With
673     End With
674     End With
675     End With
676     End With
677     End With
678     End With
679     End With
680     End With
681     End With
682     End With
683     End With
684     End With
685     End With
686     End With
687     End With
688     End With
689     End With
690     End With
691     End With
692     End With
693     End With
694     End With
695     End With
696     End With
697     End With
698     End With
699     End With
700     End With
701     End With
702     End With
703     End With
704     End With
705     End With
706     End With
707     End With
708     End With
709     End With
710     End With
711     End With
712     End With
713     End With
714     End With
715     End With
716     End With
717     End With
718     End With
719     End With
720     End With
721     End With
722     End With
723     End With
724     End With
725     End With
726     End With
727     End With
728     End With
729     End With
730     End With
731     End With
732     End With
733     End With
734     End With
735     End With
736     End With
737     End With
738     End With
739     End With
740     End With
741     End With
742     End With
743     End With
744     End With
745     End With
746     End With
747     End With
748     End With
749     End With
750     End With
751     End With
752     End With
753     End With
754     End With
755     End With
756     End With
757     End With
758     End With
759     End With
760     End With
761     End With
762     End With
763     End With
764     End With
765     End With
766     End With
767     End With
768     End With
769     End With
770     End With
771     End With
772     End With
773     End With
774     End With
775     End With
776     End With
777     End With
778     End With
779     End With
780     End With
781     End With
782     End With
783     End With
784     End With
785     End With
786     End With
787     End With
788     End With
789     End With
790     End With
791     End With
792     End With
793     End With
794     End With
795     End With
796     End With
797     End With
798     End With
799     End With
800     End With
801     End With
802     End With
803     End With
804     End With
805     End With
806     End With
807     End With
808     End With
809     End With
810     End With
811     End With
812     End With
813     End With
814     End With
815     End With
816     End With
817     End With
818     End With
819     End With
820     End With
821     End With
822     End With
823     End With
824     End With
825     End With
826     End With
827     End With
828     End With
829     End With
830     End With
831     End With
832     End With
833     End With
834     End With
835     End With
836     End With
837     End With
838     End With
839     End With
840     End With
841     End With
842     End With
843     End With
844     End With
845     End With
846     End With
847     End With
848     End With
849     End With
850     End With
851     End With
852     End With
853     End With
854     End With
855     End With
856     End With
857     End With
858     End With
859     End With
860     End With
861     End With
862     End With
863     End With
864     End With
865     End With
866     End With
867     End With
868     End With
869     End With
870     End With
871     End With
872     End With
873     End With
874     End With
875     End With
876     End With
877     End With
878     End With
879     End With
880     End With
881     End With
882     End With
883     End With
884     End With
885     End With
886     End With
887     End With
888     End With
889     End With
890     End With
891     End With
892     End With
893     End With
894     End With
895     End With
896     End With
897     End With
898     End With
899     End With
900     End With
901     End With
902     End With
903     End With
904     End With
905     End With
906     End With
907     End With
908     End With
909     End With
910     End With
911     End With
912     End With
913     End With
914     End With
915     End With
916     End With
917     End With
918     End With
919     End With
920     End With
921     End With
922     End With
923     End With
924     End With
925     End With
926     End With
927     End With
928     End With
929     End With
930     End With
931     End With
932     End With
933     End With
934     End With
935     End With
936     End With
937     End With
938     End With
939     End With
940     End With
941     End With
942     End With
943     End With
944     End With
945     End With
946     End With
947     End With
948     End With
949     End With
950     End With
951     End With
952     End With
953     End With
954     End With
955     End With
956     End With
957     End With
958     End With
959     End With
960     End With
961     End With
962     End With
963     End With
964     End With
965     End With
966     End With
967     End With
968     End With
969     End With
970     End With
971     End With
972     End With
973     End With
974     End With
975     End With
976     End With
977     End With
978     End With
979     End With
980     End With
981     End With
982     End With
983     End With
984     End With
985     End With
986     End With
987     End With
988     End With
989     End With
990     End With
991     End With
992     End With
993     End With
994     End With
995     End With
996     End With
997     End With
998     End With
999     End With
1000    End With

```

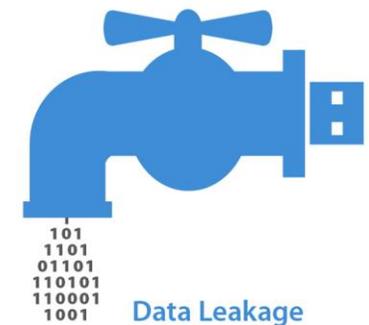


# XSSI in the Wild<sub>[USENIX15]</sub>

- Conducted a study of 150 highest-ranked sites with logins
  - sites for which we could create a login (not banks, for example)

	Domains	Exploitable
Dynamic scripts	49	40
Unique identifier	34	28
Other personal data	15	11
CSRF / auth tokens	7	4

- Several high impact flaws
  - leaked credit card info on my own bank
  - reading senders and subjects of emails
  - account hijacking for file hosting service



# Preventing XSSI

- Scripts must not be loadable from other origins
  - referrer checking (recall the problems associated with that)
  - use of secret tokens (similar to CSRF)
- Only provide code in scripts, use provisioning service for data
  - use XHR to retrieve data
  - easily protectable by SOP or CORS
- Use inline scripts only
  - with CSP nonces, even possible to use with CSP
  - can not be included remotely, hence data is secure there



# The Great Cannon



# Including third-party resources on the Web



```
← → × http://cnn.com  
  
<html>  
....  
<script src="//googletagmanager.com/tag.js">  
</script>  
...  
</html>  
  
var tags = "cnn.com";  
document.write("Doing tagging stuff here");  
// ...
```



# Including third-party resources on the Web (with MitM)

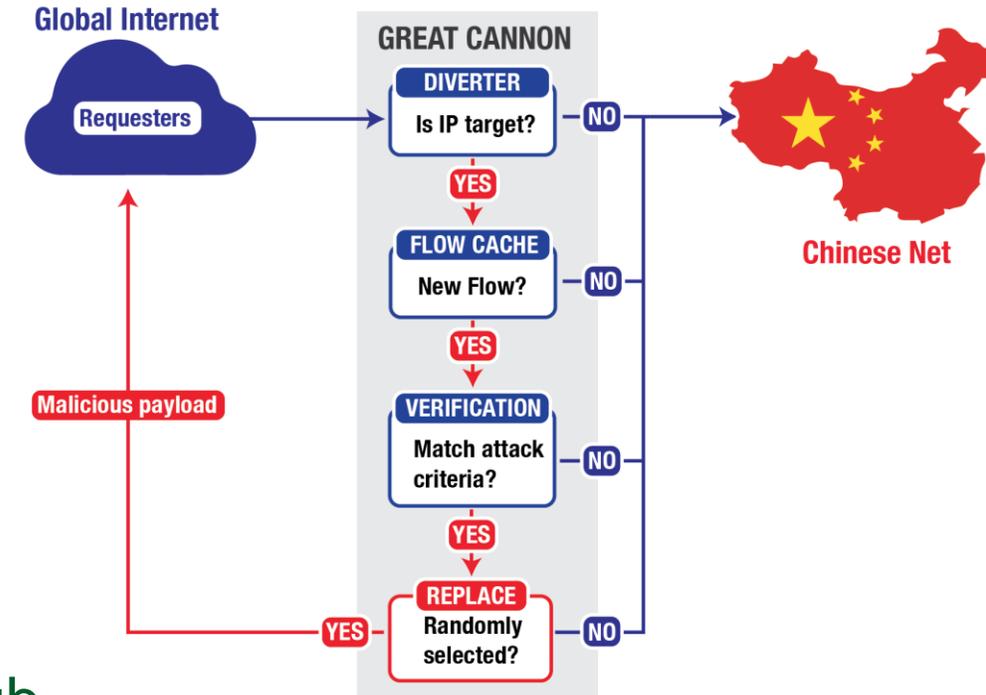


```
← → × http://cnn.com  
  
<html>  
....  
<script src="//googletagmanager.com/tag.js">  
</script>  
...  
</html>  
  
var target = "http://github.com"  
var x = new XMLHttpRequest();  
x.open("GET", target);  
// ...
```



# The Great Cannon

- China already has a powerful firewall
  - "The Great Firewall"
  - drops unwanted connections (e.g. NY Times)
- Mirror sites exists for blocked sites
  - e.g., [greatfire.org](http://greatfire.org) and several GitHub repos
- Great Cannon injected JavaScript into content from, e.g., [baidu.com](http://baidu.com)
  - millions of users opened connections to GitHub, New York Times, [greatfire.org](http://greatfire.org)
- Massively Distributed Denial of Service



<https://citizenlab.ca/2015/04/chinas-great-cannon/>

# Subresource Integrity (SRI)

- To thwart such injection attacks, SRI was proposed
- Use cryptographic hash of remote resource
  - for scripts and style sheets
  - if hash does not match, resource is ignored

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"  
integrity="sha384-R4/ztc4ZlRqWjqIuvf6RX5yb/v90qNGx6fS48N0tRxiGkqveZETq72KgDVJCp2TC"  
crossorigin="anonymous"></script>
```

- Protects against malicious CDNs/MitM attackers
  - also allows to pin to a specific version of third-party libraries

```
<script>window.jQuery || /* reload from own domain here */;</script>
```

# Subresource Integrity (SRI)

- SRI resources must be CORS-enabled
  - otherwise, SRI could be used to test remote resource for certain content
- Integrity attribute can have multiple values
  - Only strongest hash is used

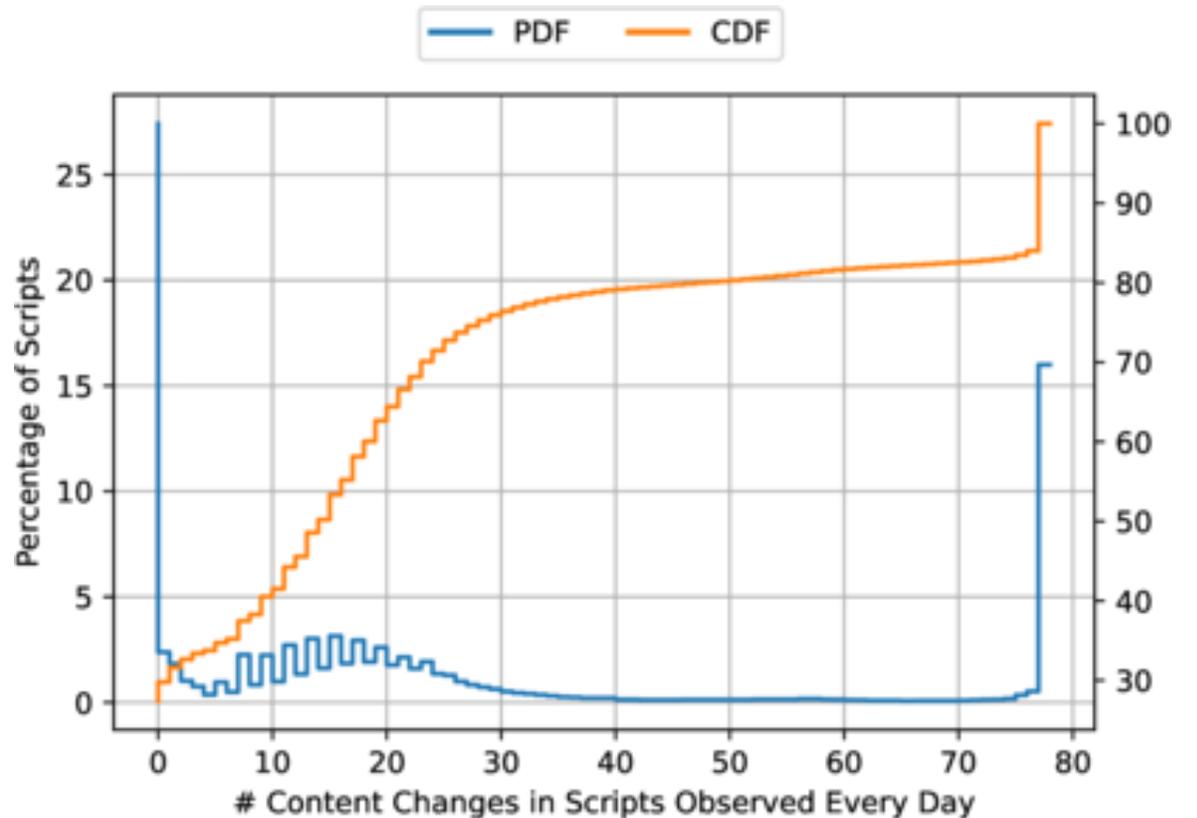
```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"  
integrity="sha384-R4/ztc4ZlRqWjqIuvf6RX5yb/v90qNGx6fS48N0tRxiGkqveZETq72KgDVJcP2TC sha256-  
8WqyJLuWKRbVhxXIL1jBDD7SDxU936oZkCnxQbWwJVw="<br>  
crossorigin="anonymous"></script>
```

- Multiple same-strength hashes are allowed but rarely used

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"  
integrity="sha256-t1X5SBfMY4/0kYdt8H1CP/90Gg0i1G6U9UnjC6AVYHA=  
sha256-8WqyJLuWKRbVhxXIL1jBDD7SDxU936oZkCnxQbWwJVw="<br>  
crossorigin="anonymous"></script>
```

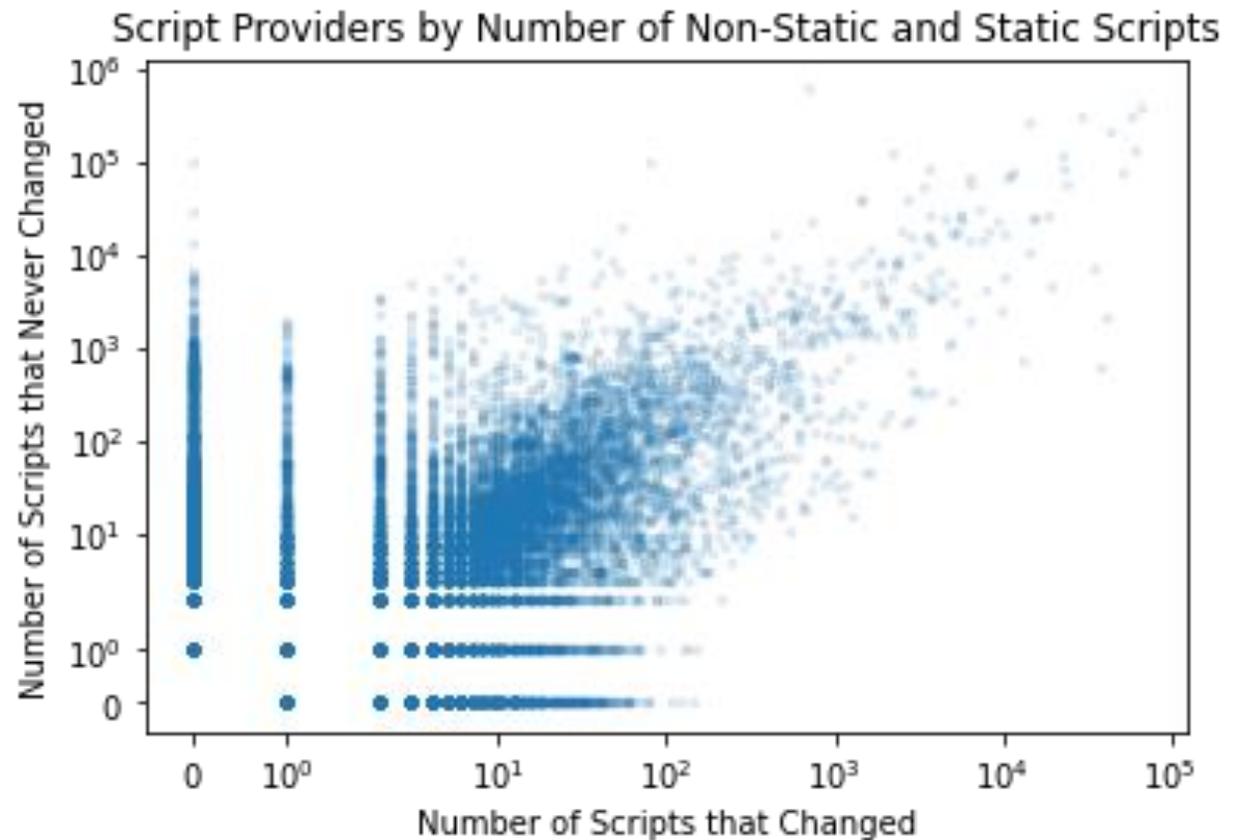
# Subresource Integrity (SRI) [WWW'23]

- SRI is applicable to static scripts, or scripts that rarely change
- What is the fraction of scripts that remain static?
  - 27% of scripts never changed
  - 16% of scripts changed every day



# Subresource Integrity (SRI) [WWW'23]

- SRI needs to be applied to every script from each third party
- How many third-party providers serve only static scripts?
  - 44% of providers serve only static scripts ( $x=0$ )
  - 24% of providers serve only changing scripts ( $y=0$ )
  - 32% of providers serve a mix of both



# Sandboxing Content



# Multi-origin Web applications

- Modern Web applications use code from multiple origins
  - Analytics
  - Advertisement
  - Maps
  - ....
- Even framed content may, e.g., open a popup
  - or redirect the parent frame
- Necessity for control privileges of included content arises
  - putting everybody in their own little sandbox



# Sandboxing iframes

- Limits iframe's ability to conduct certain actions
  - e.g., disable JavaScript, putting them in an isolated origin
- Just adding sandbox to the iframe will restrict everything
  - rights have to be granted explicitly
    - `allow-forms`: allows for form submission in iframe
    - `allow-popups`: enables popups
    - `allow-pointer-lock`: enable PointerLock API to get raw mouse movements
    - `allow-scripts`: enable scripting
    - `allow-same-origin`: enable origin of included page, not isolated one
    - `allow-top-navigation`: enables navigating the top frame

# Sandbox usage examples

```

<textarea id='code'></textarea>
<button id='safe'>eval() in a sandboxed frame.</button>
<iframe sandbox='allow-scripts' id='sbox' src='frame.html'>
</iframe>

<script>
  function evaluate() {
    sandboxed.contentWindow.postMessage(code.value, '*');
  }
  safe.addEventListener('click', evaluate);

  window.addEventListener('message', function (e) {
    if (e.origin === "null" && e.source === sbox.contentWindow)
      alert('Result: ' + e.data);
  });
</script>

```

Parent page

```

<script>
  window.addEventListener('message', function (e) {
    if (e.origin !== "https://main.com") {
      return
    }
    var mainWindow = e.source;
    var result = '';

    try {
      result = eval(e.data);
    } catch (e) {
      result = 'eval() threw an exception.';
    }
    mainWindow.postMessage(result, e.origin);
  });
</script>

```

frame.html

# Determining least privilege

- Example: tweet button
  - opens popup window
  - submit a form
  - sends authenticated request to [twitter.com](https://twitter.com) (using and accesses document.cookie)
- Requires four permissions
  - allow-popups (well, it opens a popup..)
  - allow-forms (well, it is a form)
  - allow-same-origin (JavaScript needs access to cookies)
  - allow-scripts (not too much of a surprise)

# Determining least privilege

- Example: tweet button
  - opens popup window
  - submit a form
  - sends authenticated request to [twitter.com](https://twitter.com) (using and accesses document cookie)

- Requires for

```
<iframe sandbox="allow-same-origin allow-scripts allow-popups allow-forms"
  src="https://platform.twitter.com/widgets/tweet_button.html"
  style="border: 0; width:130px; height:20px;"></iframe>
```

- allow-popups (well, it opens a popup..)
- allow-forms (well, it is a form)
- allow-same-origin (JavaScript needs access to cookies)
- allow-scripts (not too much of a surprise)

# Sandboxing with CSP

- Limits the entire page's ability to conduct certain actions
  - as if the page were loaded in an iframe with the sandbox attribute
- What's the difference?
  - Functionally the same, but they differ in scope
  - CSP sandboxing applies to the entire page
  - iframe sandboxing applies only to that iframe
- When would you use one over the other?
  - e.g., using CSP sandboxing on a page that is an archived instance of a web vulnerability that triggers drive-by downloads

# Summary

4

## Forcing browser to perform an action for the attacker

```

<form method="POST" action="https://acmebank.com/transfer" id="transfer">
  <input type="hidden" name="act-to" value="987-654-3210">
  <input type="hidden" name="amount" value="100000">
</form>
<script>
transfer.submit()
</script>

```

35

## Subresource Integrity (SRI)

- To thwart such injection attacks, SRI was proposed
- Use cryptographic hash of remote resource
  - for scripts and style sheets
  - if hash does not match, resource is ignored

```

<script src="https://code.jquery.com/jquery-2.1.4.min.js" integrity="sha384-R4/ztc421RqWjqIuvf6RX5yb/v90qNGx6f548N0tRxIGkqveZETq72KgDVCp2TC" crossorigin="anonymous"></script>

```

- Protects against malicious CDNs/MitM attackers
  - also allows to pin to a specific version of third-party libraries

```

<script>window.jQuery || /* reload from own domain here */;</script>

```

24

## Cross-Site Scripting Inclusion (XSSI)

- Regular scripts may also be dynamically generated
  - We cannot read the source code, but can observe side-effects

```

<script>
// Register global function
function show_contacts(contacts) {
  // Steal data here
}
</script>
<script src="//gmail.com/contacts.js">
</script>

```

```

show_contacts([
  ['ct', 'John Doe', 'foo@gmail.com'],
  ['ct', 'Jane Doe', 'bar@gmail.com']
])

```

40

## Sandbox usage examples

```

<textarea id='code'></textarea>
<button id='safe'>eval() in a sandboxed frame.</button>
<iframe sandbox='allow-scripts' id='sbox' src='frame.html'>
</iframe>
<script>
function evaluate() {
  sandboxed.contentWindow.postMessage(code.value, '*');
}
safe.addEventListener('click', evaluate);

```

```

<script>
window.addEventListener('message', function (e) {
  if (e.origin !== 'https://main.com') {
    return
  }
  var mainWindow = e.source;
  var result = '';
  try {
    result = eval(e.data);
  } catch (e) {
    result = 'eval() threw an exception.';
  }
  mainWindow.postMessage(result, e.origin);
});
</script>

```

Parent page

frame.html

<https://www.html5rocks.com/en/static/demos/sandbox/index.html>

# Credits

- Original slide deck by Ben Stock
- Modified by Nick Nikiforakis