JARED M. SMITH

HACKING YOUR SYSTEMS BEFORE THEY DO FIRST

ABOUT ME

- Security Researcher and Software Engineer Intern at ORNL in the Cyber Warfare Research Team
- Formerly Software Security Engineer Intern at Cisco System's Advanced Security Initiatives Group
- Senior in Honors Computer Science at UTK
- President and Co-Founder of HackUTK
- Technical Consultant for several startups (cybersecurity, media, and social spaces)

A (SLIGHTLY SATIRICAL)
DAY IN THE LIFE OF A DEV

Lead Developer: Boss, we've been building this application for weeks. We think we're finally at 1.0 and the client likes the preliminary results. Even all of our continuous integration tests are passing! Can we launch it?!

Manager: Let me check with the other managers, PM's, and the CTO and I'll get back to you ASAP.

[at all-hands meeting the following day]

CTO: We've been waiting for this day for a long time…LAUNCH IT NOW!

Lead Developer: *[…scurrying to keyboard…types `git push origin release`…anxious waiting…terminal returns `TESTS PASSED. V. 1.0 DEPLOYED TO RELEASE BRANCH`…]*

Slack Bot: 🎉🎉🎉

6 MONTHS LATER...

[…massive leak of credit card numbers and plaintext passwords…hackers breached internal systems as well]

Slack Bot: SEND 1000 BITCOINS TO WALLET 1JrVSjiqq6MA1dmnu5NXHvYcqjMUNi8je9 TO GET YOUR BOT BACK

"If you spend more on coffee than on IT security you will be hacked. What's more, you deserve to be hacked."

–RICHARD CLARKE

FORMER NATIONAL COORDINATOR FOR SECURITY, INFRASTRUCTURE PROTECTION AND COUNTER-TERRORISM FOR THE UNITED STATES

SO IT'S A PROBLEM

World's Biggest Data Breaches

Selected losses greater than 30,000 records

(updated 5th Feb 2015)

interesting story

YEAR | *BUBBLE COLOUR* YEAR | METHOD OF LEAK | *BUBBLE SIZE* NO OF RECORDS STOLEN | DATA SENSITIVITY | HIDE FILTER

latest

2014

Anthem
80, 000, 000

Home Depot
56, 000, 000

Japan Airlines

JP Morgan Chase
76, 000, 000

Dominios Pizzas (France)

Community Health Services

European Central Bank

Ebay
145, 000, 000

Mozilla

MacRumours.c

Korea Credit Bureau

NASDAQ

New York Taxis

Neiman Marcus

D&B, Altegrity

AOL
2, 400, 000

Florida Department of Juvenile Justice

SnapChat

Living Social

Citigroup

Advocate

South Africa

UbiSoft

ssndob.ms

Staples

UPS

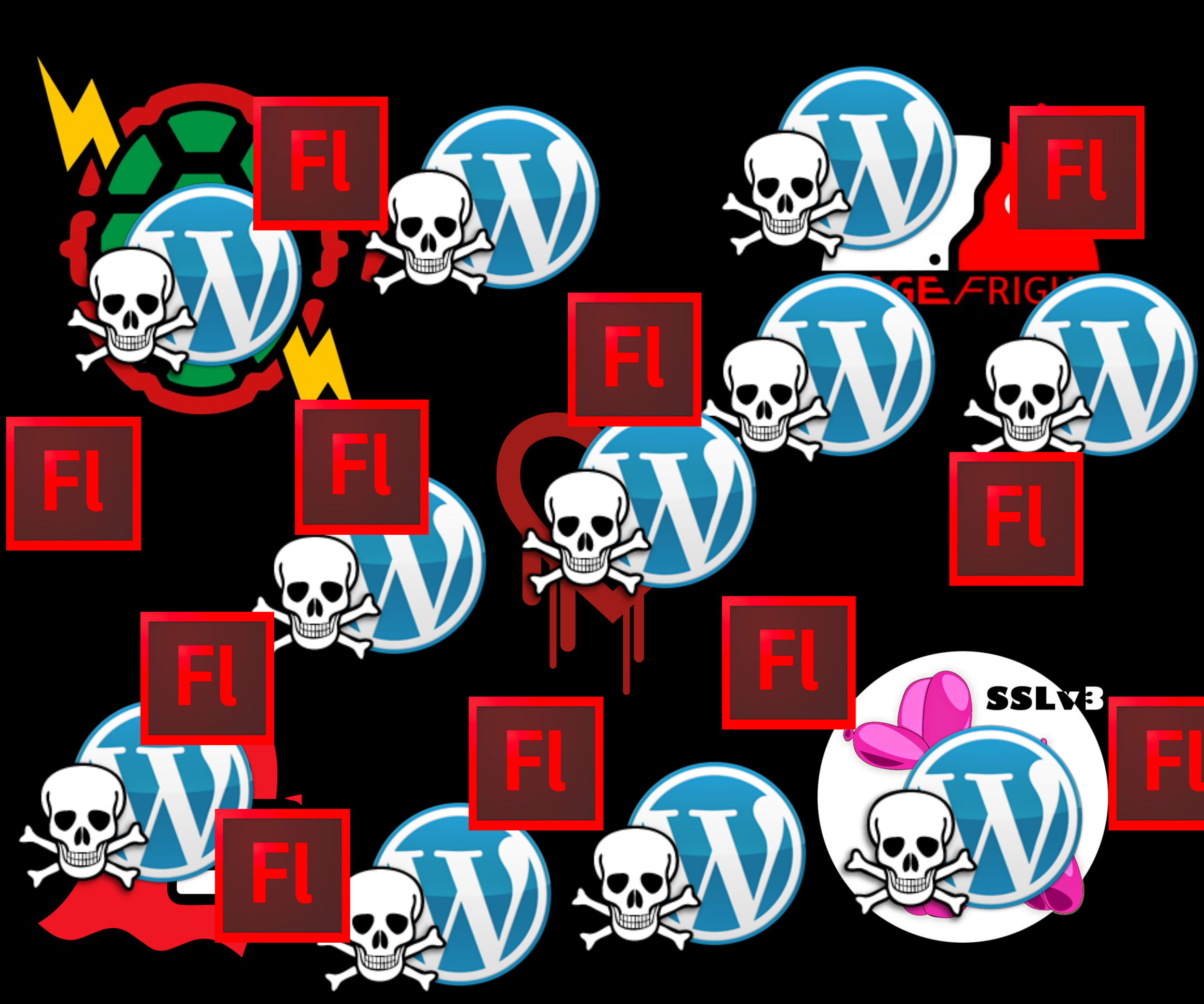
Filter by...

ORGANISATION

- (●) all
- ( ) academic
- ( ) energy
- ( ) financial
- ( ) gaming
- ( ) government
- ( ) healthcare
- ( ) media
- ( ) military
- ( ) retail
- ( ) tech
- ( ) telecoms
- ( ) transport
- ( ) web

METHOD OF LEAK

- (●) all
- ( ) accidentally published
- ( ) hacked
- ( ) inside job
- ( ) lost / stolen computer
- ( ) lost / stolen media
- ( ) poor security


SSLv3

IT'S INEVITABLE

- Because **breaches will happen**
- Zero-days will be **discovered** (i.e. heartbleed, stagefright, etc.)
- And you likely **won't see it coming** or **know how it even happened**

AND IT'S HARD

DOMAINS OF SECURITY

- Web
- Networks
- Vehicles
- Mobile
- Internet of Things (IoT)
- Embedded
- SCADA (Industrial Systems)
- and more…

VULNERABILITIES CAN BE EVERYWHERE

TCP/IP Model	Protocols and Services	OSI Model
Application	HTTP, FTP, Telnet, NTP, DHCP, PING, DNS, BGP, JSON-RPC, SOAP, Exchange ActiveSync, IMAP, LDAP, POP, RTP, SIP, VOIP, SMTP, SSH, TLS/SSL, WebSockets, DDP, Tor, STOMP, RDP, NFS, MQTT, IRC BitTorrent, Bitcoin, SOCKS, SPDY, ASCII, UTF-8, MIME, AFP, PFIF, XML-RPC	Application
		Presentation
		Session
Transport	TCP, UDP, PPTP	Transport
Network	IPv4, IPv6, ICMP, IGMP, IPSEC	Network
Network Interface	MAC (Ethernet, DSL, ISDN, FDDI), 802.11 (Wi-Fi), ARP, ATM, LLDP, NDP, PPP, OSPF, L2TP	Data Link
		Physical

BUT THERE IS HOPE

THE HACKER MINDSET

WHAT'S A VULNERABILITY?

- When you make the application do something that it's not supposed to do

- To find vulnerabilities, you have to **understand** the application

- That's **GREAT** for us developers!

KNOW THE APPLICATION

- What's the intended functionality?
- What's the intended behavior?
- What does the application use as input?
- What does the application produce as output?

IS IT A VULNERABILITY?

- You find that unauthenticated users can edit page content
- Vulnerability?
  - cnn.com - YES
  - wikipedia.org - NO

ATTACKING IS A PROCESS

- **Step 1:** Reconnaissance
- **Step 2:** Develop vulnerability hypothesis
- **Step 3:** Test vulnerability hypothesis
- **Step 4:** Develop exploit
- **Step 5:** Profit

INJECTION VECTORS

- **Understand ALL input to the application**

- Functionality from other websites
- Emails
- Form fields
- Web Sockets

- Query parameters
- URL path
- PUT/POST parameters
- Cookies
- Headers (Referer header)

UNDERSTAND DATA FLOW

- How does the input data flow through the program?
  - **Example:** Data on page X is displayed on page Y and used to calculate the result of page Z
- How does the output of a page flow through the program?
  - **Example:** the result of a calculation used as part of a tweet

KNOW YOUR BASICS

LEGEND

A blue box with text inside means it specifically refers to JavaScript

Look for light blue words for pointers to concepts important for mitigating specific vulnerabilities

THE BASICS

- Injection
- Broken authentication and sessions
- Cross-Site Scripting (XSS)
- Exposing internal implementation details
- Bad Config
- Exposing/Leaking Sensitive Data
- Not validating access to functionality on the server side
- Cross-Site Request Forgery (CSRF)

THE BASICS

- Using other people's vulnerable code
- Redirecting/Forwarding using unvalidated input
- Over/Under posting
- File Inclusion
- Password management
- Remote Code Execution
- and many more…

DIVING DEEPER

CROSS-SITE SCRIPTING

- Run malicious scripts (JavaScript) in victim's browser and targeted application
- Successful XSS can steal user information (cookies, session IDs, etc.), redirect them to malicious websites, act on users behalf, the list goes on and on…

MITIGATION

- Validate and sanitize all user input!
- Encode output for specific contexts
- Set HTTPOnly cookie flags on cookies that the client doesn't need to access
- Use Content Security Policy (CSP) headers to only allow certain client side resources to be used by the applications
- Apply encoding on both client and server side to prevent DOM based XSS attacks

  - Swig Autoescape mode (http://paularmstrong.github.io/swig/docs/api/#SwigOpts)

CSRF

- Forces the end user to execute unwanted actions on a web application in which they're currently authenticated
- Attackers usually target **state-changing** requests
- Possible to store CSRF attacks in IMG or IFRAME tags in fields that accept HTML

MITIGATION

- Include a random, unpredictable token in requests
- Add tokens to requests which mutate state

- https://github.com/expressjs/csurf
- **BE SURE** to add this middleware before any other middleware that modifies requests

INJECTION

- Occurs when untrusted data is sent to an interpreter as part of a command or query
- Injection of malicious code into programmer logic, through SQL, eval statements, OS instructions, LDAP, etc…
- In JavaScript, manifests itself often when using eval(), setTimeout(), setInterval(), Function(), and database queries (you too MongoDB!)

MITIGATION

- Validate all user input on the server side before processing it!

- Do not use eval() or other similar commands to parse user inputs
- When parsing JSON, use appropriate parsing methods, NOT eval()
- Include “use strict” at the beginning of functions

PASSWORD MANAGEMENT

- Passwords are often handled incorrectly
- Passwords are left in **plaintext** or **encrypted**
- **Example:**
  - 2012 LinkedIn breach
  - Passwords were hashed, but failed to use a salt
  - Easily cracked using existing rainbow tables

MITIGATION

- Use bcrypt for hashing passwords with salts
- Enforce strong passwords!

- JavaScript bcrypt implementation - https://github.com/ncb000gt/node.bcrypt.js/

USING OTHER PEOPLE'S CODE

- Using 3rd party libraries and/or including other peoples code can lead to vulnerabilities
- Some things you probably use a lot:
  - Google Analytics
  - Bootstrap Javascript files
  - Lo-dash for common JavaScript functions
- **Example:** early internet page counter scripts

MITIGATION

- Don’t give trust blindly
- Know what you’re including in your own code and applications
- Review the code you include for trustworthiness!

BROKEN AUTHENTICATION AND SESSION MANAGEMENT

- Attacker uses leaks or flaws in the authentication or session management functions to impersonate other users

- Developers often build custom authentication and session management schemes

  - This is **very** hard

  - Often have flaws in implementation

MITIGATION

- Protect user authentication credentials
- Don’t expose session ID’s in URL
- Session ID’s should timeout
- Recreate session ID’s after successful login
- Don’t send session ID’s and credentials over unencrypted connections (Use SSL/TLS)
- Make sure that a user is authorized to use a resource even if already authenticated (permissions)

MITIGATION

- Enforce permissions: https://github.com/OptimalBits/node_acl

- Secure requests with headers: https://github.com/helmetjs/helmet

- Authentication: http://passportjs.org/docs/authenticate

OVER/UNDER POSTING

- Occurs when the application allows restricted parts of entities to be updated without authorization
- Returning entire entities and updating them on a whim is

bad

```
1  public User GetUser(id) {
2      var user = myOrm.getUser(id);
3
4      return user;
5  }
6
7  public void PostUser(user) {
8      myOrm.SaveUser(user);
9  }
```

check your privilege

MITIGATION

- Follow the Principle of Least Privilege
- Only return the parts of entities that are required for any given request/response
- Only update the parts of entities that are supposed to be updatable
    - That means don’t allow regular users to change *isAdmin* on an entity!

SENSITIVE DATA EXPOSURE

- Don’t expose sensitive data! It’s a no brainer!
- Users put their trust in you, so don’t break it

MITIGATION

- Use SSL/TLS (https over http)
- Encrypt all sensitive data at rest and in transit
- Don’t store user’s sensitive data unnecessarily
- Use strong crypto and strong key management
- Disable caching on forms that store sensitive data

DEMO

OUR SETUP

https://github.com/OWASP/NodeGoat

https://github.com/zaproxy/zaproxy

WRAPPING UP

GET OUT THERE!

Knoxville BSides 2016

HackUTK (for utk students)

Knoxville Security Meetup

DEFCON 24

CONTACT

- **Jared M. Smith**
- **Twitter:** twitter.com/jaredthecoder
- **Website:** jaredmichaelsmith.com
- **LinkedIn:** linkedin.com/in/jaredmichaelsmith
- **Email:** jared@jaredsmith.io

CREDITS

- Open source and security communities
- (current) ORNL and (past) Cisco (for providing me the opportunity to do this stuff on a daily basis)
- KnoxDevs (for their encouragement and support of the local dev community)
- Adam Doupé - Professor at Arizona State University (inspiration for Hacker Mindset slides)
- Josh Carroll and Chris Lamm (support throughout the process of preparing for this talk)
- Joseph Connor and Kaleigh Veca (reviewing my slides)

GENERAL RESOURCES

- OWASP Top 10: https://www.owasp.org/index.php/Top_10_2013-Top_10
- Kali Linux: https://www.kali.org/downloads/
- OWASP: https://www.owasp.org/
- Websites to Hack Yourself: https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project
- Lists:
    - https://github.com/infoslack/awesome-web-hacking
    - http://sectools.org/
- Conferences:
    - https://bsidesknoxville.com/
    - https://www.defcon.org/
- Local groups:
    - https://hackutk.com
    - http://www.meetup.com/defcon865/
- Books:
    - http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470
    - http://www.amazon.com/Hacking-The-Art-Exploitation-Edition/dp/1593271441

OTHER SCANNERS

- Free/OSS:
  - Wapiti: http://wapiti.sourceforge.net/
  - Nikto: https://cirt.net/Nikto2
  - w3af: http://w3af.org/
- Proprietary:
  - BurpSuite Proxy and Scanner: https://portswigger.net/burp/
  - NetSparker: https://www.netsparker.com/web-vulnerability-scanner/
  - Nessus by Tenable: https://www.tenable.com/products/nessus-vulnerability-scanner

JAVASCRIPT SECURITY RESOURCES

- http://nodegoat.herokuapp.com/tutorial
- https://blog.risingstack.com/node-js-security-checklist/
- https://nodesecurity.io/resources
- http://expressjs.com/en/advanced/best-practice-security.html
- http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html
- https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet
- Books:
    - https://pragprog.com/book/kdnodesec/secure-your-node-js-web-application

"Companies spend millions of dollars on firewalls, encryption, and secure access devices, and it's money wasted, because none of these measures address the weakest link in the security chain."

–KEVIN MITNICK

COMPUTER SECURITY CONSULTANT, AUTHOR, AND HACKER

BACKUP SLIDES

CSRF - EXAMPLE

- Executed through GET or POST requests
  - Original: GET http://bank.com/transfer.do?acct=JOSH&amount=100 HTTP/1.1
  - Jared changes to: http://bank.com/transfer.do?acct=JARED&amount=1000000
- Jared constructs a malicious link:
  - View my Pictures!
  - <img src="http://bank.com/transfer.do?acct=MARIA&amount=100000" width="0" height="0" border=“0">
- Gets a user to click it with social engineering or XSS

INJECTION - EXAMPLE

Vulnerable code behind the scenes

statement = "SELECT * FROM users WHERE name = '" + userName + “';"

Attacker inserts this in query parameter/form field/etc.

' OR ‘1’='1

Which renders these...

SELECT * FROM users WHERE name = '' OR ‘1'='1';