

Security

<https://outflux.net/slides/2015/osu-devops.pdf>



DevOps Bootcamp, OSU, Feb 2015

Kees Cook <kees@outflux.net>

(pronounced "Case")

Who is this guy?

- Fun:

- DefCon CTF
 - team won in 2006 & 2007
- Debian
- Ubuntu



- Jobs:

- OSDL (proto Linux Foundation)
- Canonical (Ubuntu Security)
- Google (Chrome OS Security)



Agenda

- Supplement last week's topics
 - Security updates
 - Threat models
 - Finding vulnerabilities
- Secure communication
 - SSL model
 - Attackers
 - Exercise



Security updates: do it

- Keeping systems secure:
 - Follow principles of least privilege
 - Promptly install latest security updates
 - Everything else



Security updates: stay educated

- Find out where to learn more
 - Find a mailing list
 - e.g. Ubuntu's security update mailing list
<https://lists.ubuntu.com/mailman/listinfo/ubuntu-security-announce>
 - Find a blog
 - e.g. Chrome's release update blog
<http://googlechromereleases.blogspot.com/>



Security updates: even the kernel

- Installed a kernel update?
Reboot!
- If this is “annoying”, then your production environment is not robust enough.
- Make reboots an expected part of the design and/or schedule:
 - Load balancers?
 - Scheduled downtime?



Threat models: attribution

- Who is the attacker?
- Purpose
 - Vandalism
 - Personal gain
- Funding
 - Individuals tend not to be well funded
 - Highly funded: state actors, corporations



Threat models: targeting

- Why is the attack happening?
- Scatter shot
 - Vulnerable software version
- Personal
 - Looking for specific information



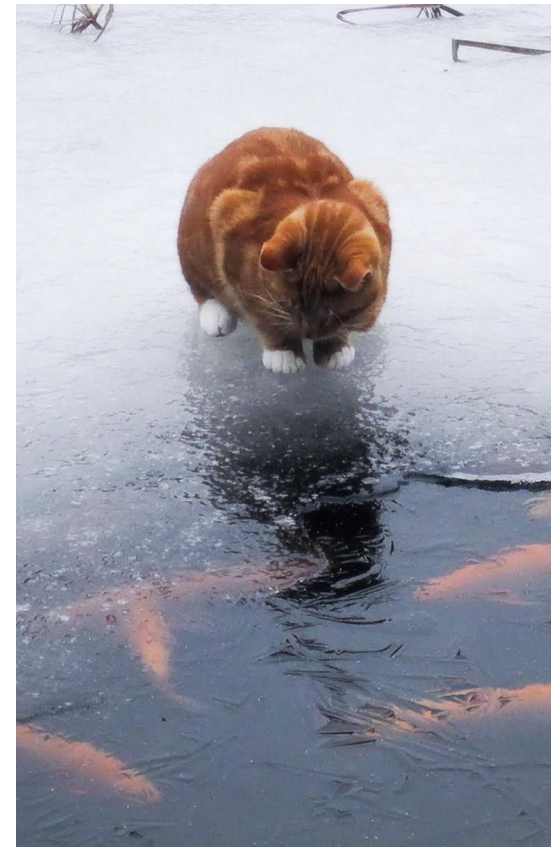
Finding vulnerabilities: timeline

- “What to **DO** if you discover a vulnerability”
 - Excellent summary on disclosure procedure
 - Keep in mind, it's your discovery
 - Communicate a desired timeline
 - Linux kernel uses max 5 working days
 - Google Project Zero uses max 90 days
 - Adobe and others have gone months, if not years



Finding vulnerabilities: go looking

- “What to do **IF** you discover a vulnerability”
 - How about *when* you discover ...
- Test everything
 - Service monitoring
 - Easy refactoring
 - Fewer bugs
 - Security flaws are just bugs, right?



Secure Communication: interception

- Most control over the server, service, etc
- Less control over the client
- Virtually zero control over communication path



Secure Communication: testing

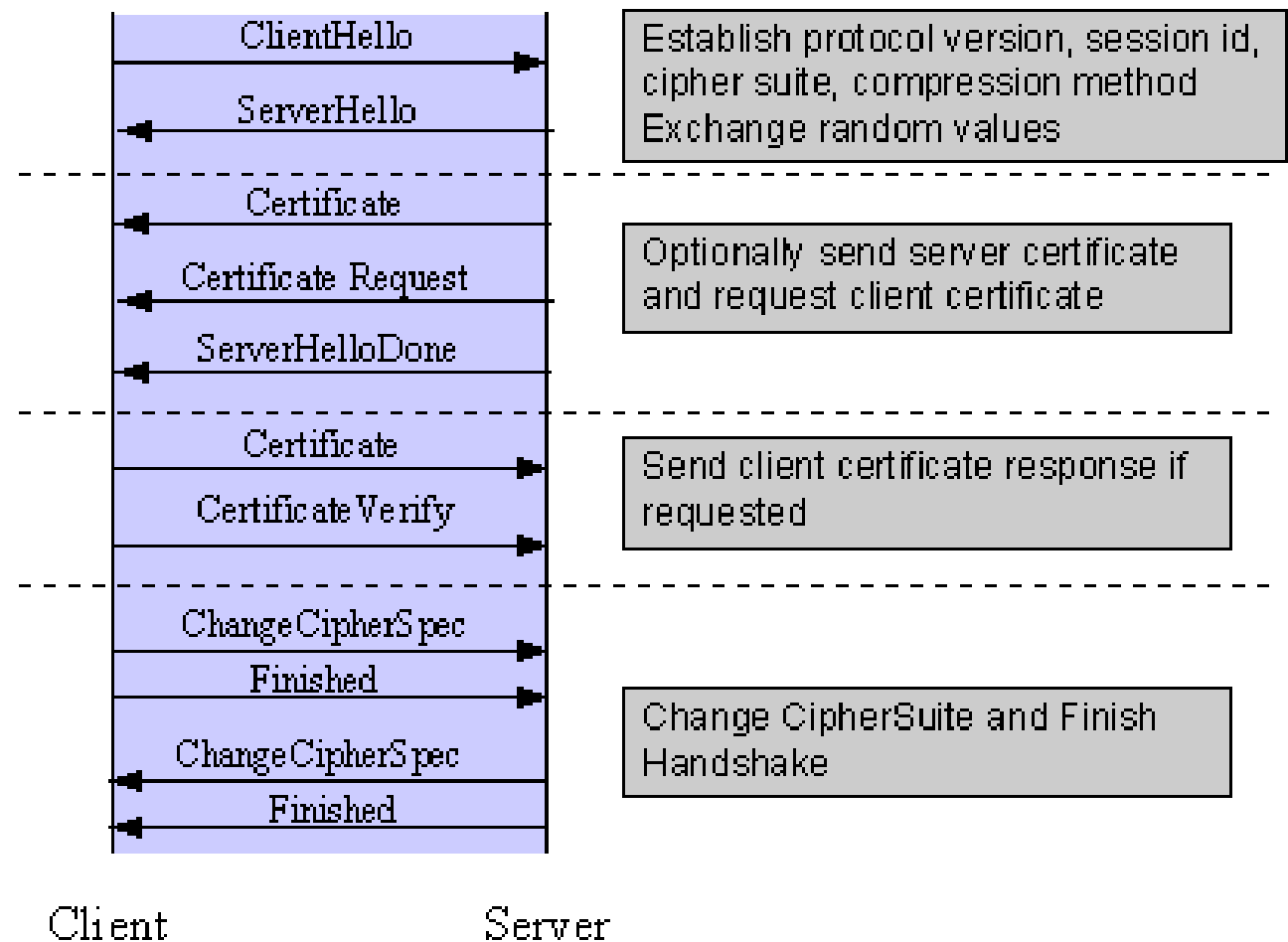
- Encryption works, but only when done right
- “Done right” requires testing



Secure Communication: https

http://httpd.apache.org/docs/2.4/ssl/ssl_intro.html

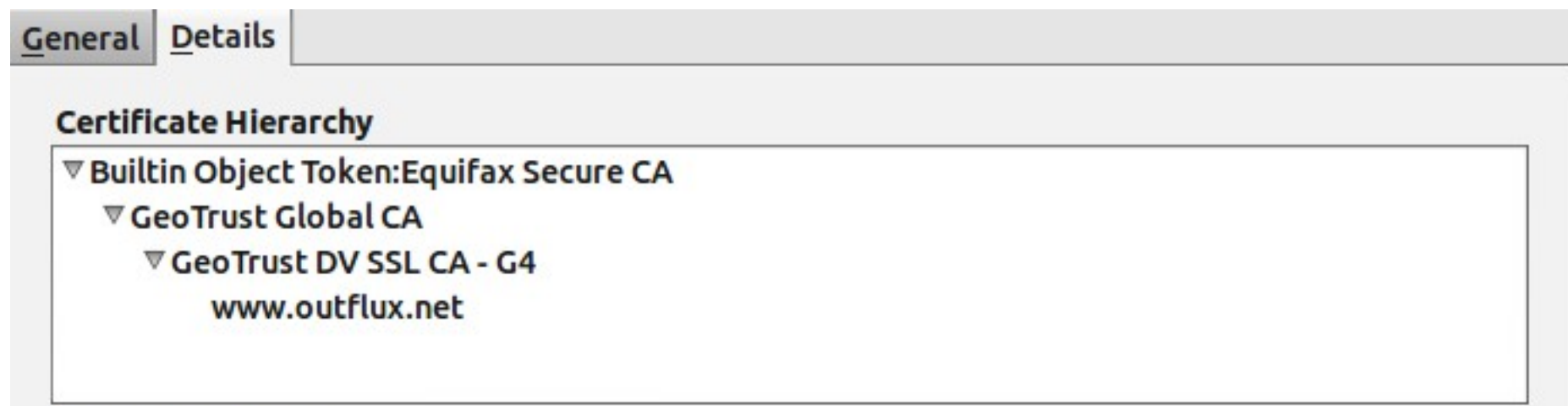
- Encryption built between TCP and HTTP
- Generally called SSL, strictly speaking it's TLS
- Trust via browser's Certificate Authority list



Secure Communication: SSL

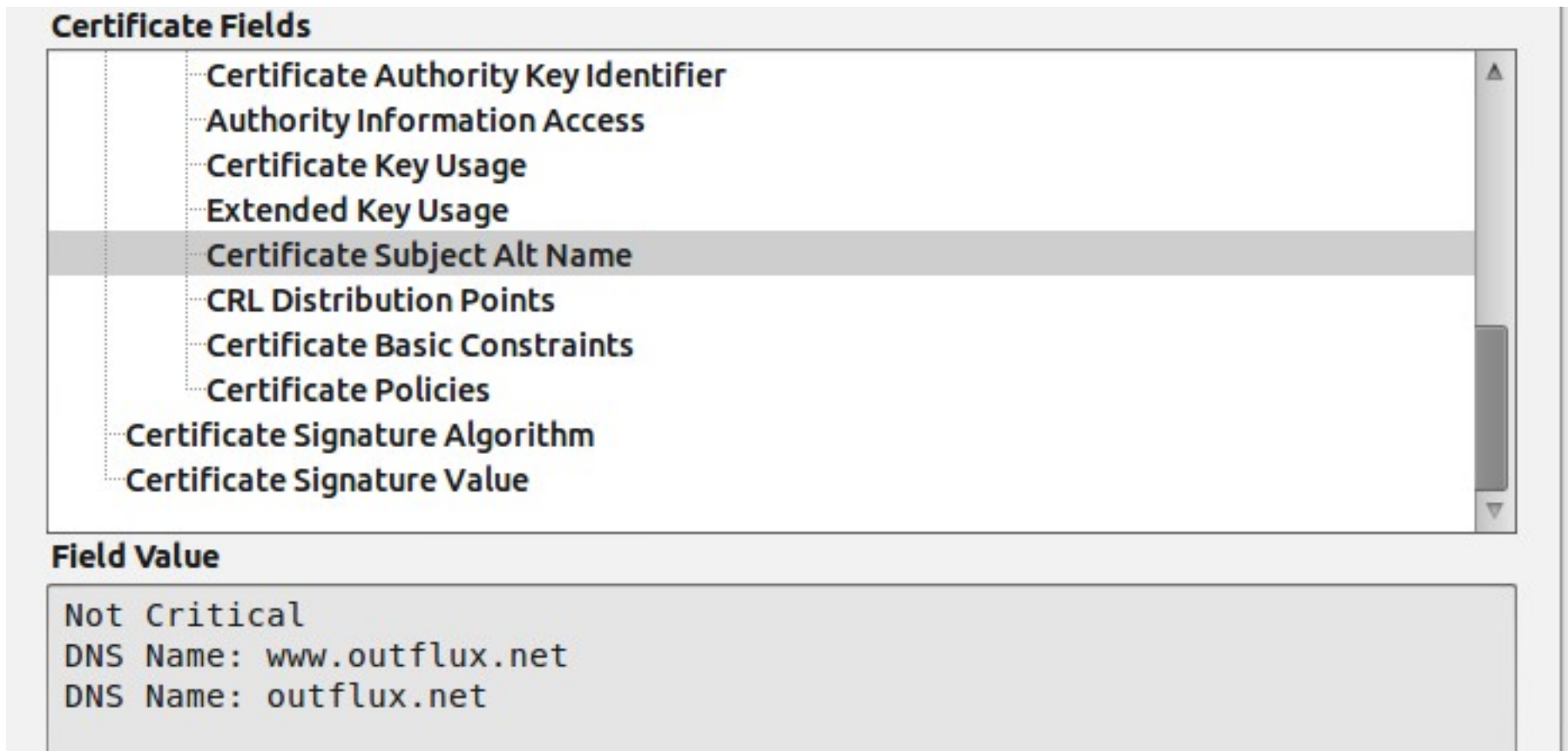
- Cryptographic chain of trust

Using <https://outflux.net/> as an example ...



Secure Communication: SSL

- Trust tied to domain names



The image shows a screenshot of a software interface displaying a list of certificate fields. The list is titled "Certificate Fields" and includes the following items:

- Certificate Authority Key Identifier
- Authority Information Access
- Certificate Key Usage
- Extended Key Usage
- Certificate Subject Alt Name** (highlighted)
- CRL Distribution Points
- Certificate Basic Constraints
- Certificate Policies
- Certificate Signature Algorithm
- Certificate Signature Value

Below the list, there is a section titled "Field Value" which contains the following text:

```
Not Critical  
DNS Name: www.outflux.net  
DNS Name: outflux.net
```

Exercise: Programmatic https

- Demo website
- Log into server environment
- Watch some network interception tools
- Write some Python to perform:
 - HTTP
 - HTTP with authentication
 - HTTPS with authentication
 - Verified HTTPS with authentication

demo website

<http://outflux.net/osu/devops/>

- User: class
- Password: omgworstpassword

environment

- Log into my EC2 instance
 - Username: class Password: osu-devops
 - ssh class@osu-devops.outflux.net
 - ECDSA key fingerprint is
ae:9b:ed:87:72:b5:0c:4d:97:ef:d6:ac:56:72:a4:65.
 - RSA: ef:e4:88:fe:16:66:7f:b3:6a:31:ed:75:dd:c8:5f:3e
- Please don't step on each other
 - mkdir your-nick-here
 - cd your-nick-here

simple HTTP fetch

- Create “devops-test.py” with an editor
- Run with: “python devops-test.py”

```
#!/usr/bin/env python
```

```
import urllib2
```

```
url = 'http://outflux.net/osu/devops/data.txt'
```

```
request = urllib2.Request(url)
```

```
response = urllib2.urlopen(request)
```

```
html = response.read()
```

```
print html,
```

simple network interception

- We can watch network traffic with tcpdump:

```
tcpdump -i eth0 -p "port 80"
```

HTTP fetch with authentication

```
--- http-noauth.py 2015-02-01 12:04:38.663740446 -0800
+++ http-auth.py 2015-02-01 12:05:08.451888157 -0800
@@ -1,8 +1,13 @@
#!/usr/bin/env python
import urllib2
+import base64

-url = 'http://outflux.net/osu/devops/data.txt'
+url = 'http://outflux.net/osu/devops/secret/data.txt'
+username = 'class'
+password = 'omgworstpassword'
request = urllib2.Request(url)
+base64string = base64.encodestring('%s:%s' % (username, password)).strip()
+request.add_header("Authorization", "Basic %s" % base64string)
response = urllib2.urlopen(request)
html = response.read()
print html,
```

decoded network interception

- We can watch decoded network traffic with wireshark's command line “tshark” tool:

```
tshark -i eth0 -Y "http or ssl" \  
-z "proto,colinfo,http.authbasic,http.authbasic"
```

HTTPS fetch with authentication

```
--- http-auth.py      2015-02-01 12:05:08.451888157 -0800
+++ https-auth.py    2015-02-01 12:13:02.366238172 -0800
@@ -2,7 +2,7 @@
import urllib2
import base64

-url = 'http://outflux.net/osu/devops/secret/data.txt'
+url = 'https://outflux.net/osu/devops/secret/data.txt'
username = 'class'
password = 'omgworstpassword'
request = urllib2.Request(url)
```

intercept: transparent proxy setup

- tshark not very useful any more
- I've set up a transparent MitM proxy, "mitmproxy"

```
sysctl -w net.ipv4.ip_forward=1
```

```
echo 0 | sudo tee \
```

```
  /proc/sys/net/ipv4/conf/*/send_redirects
```

```
iptables -t nat -A OUTPUT ! -o lo \
```

```
  -m owner --uid-owner 1001 -p tcp --dport 80 \
```

```
  -j REDIRECT --to-port 8080
```

```
iptables -t nat -A OUTPUT ! -o lo \
```

```
  -m owner --uid-owner 1001 -p tcp --dport 443 \
```

```
  -j REDIRECT --to-port 8080
```


intercept: mitmproxy

- Added some code to decode base64 of HTTP Basic Auth, otherwise stock:

```
mitmproxy -T -p 8080 --host \  
  --anticache -s decode_basic.py
```

Guess what? urllib2 is broken.

- Prior to Python 2.7.9 (Dec 10, 2014), there was no sane way to get urllib2 to perform correct SSL verification of Certificate Chain nor host name checking!
- To quote from the Python documentation at <https://docs.python.org/2/library/httplib.html#httplib.HTTPSConnection>

```
class httplib.HTTPSConnection ...
```

Changed in version 2.7.9: context was added.

This class now performs all the necessary certificate and hostname checks by default. To revert to the previous, unverified, behavior `ssl._create_unverified_context()` can be passed to the context parameter.

Python cURL (no verify)

```
#!/usr/bin/env python
import pycurl

url = 'https://outflux.net/osu/devops/secret/data.txt'
curl = pycurl.Curl()
curl.setopt(pycurl.SSL_VERIFYPEER, 0)
curl.setopt(pycurl.SSL_VERIFYHOST, 0)
curl.setopt(pycurl.USERPWD, 'class:omgworstpassword')
curl.setopt(pycurl.URL, url)

curl.perform()
```

Python cURL (verified)

```
--- https-auth-curl-weak.py 2015-02-01 22:24:44.820235813 -0800
+++ https-auth-curl.py 2015-02-01 06:39:07.790892077 -0800
@@ -2,8 +2,6 @@
 import pycurl

 curl = pycurl.Curl()
-curl.setopt(pycurl.SSL_VERIFYPEER, 0)
-curl.setopt(pycurl.SSL_VERIFYHOST, 0)
 curl.setopt(pycurl.USERPWD, 'class:omgworstpassword')
 curl.setopt(pycurl.URL, "https://outflux.net/osu/devops/secret/data.txt")
```

Why is traffic still decrypted?

- mitmproxy can still see traffic contents
- pycurl test script thinks there is no problem
- SSL clients need to do two things:
 - Check Certificate Trust Chain
 - Check Certificate host name



Let's look at the Certificate

```
$ echo QUIT | openssl s_client \  
-connect outflux.net:443
```

...

Certificate chain

```
0 s:/CN=www.outflux.net  
  i:/CN=mitmproxy/0=mitmproxy  
1 s:/CN=mitmproxy/0=mitmproxy  
  i:/CN=mitmproxy/0=mitmproxy
```

...

Where is the Trusted Cert list?

```
$ strace -e trace=file  
    ./https-auth-curl.py 2>&1 | \  
    grep open | egrep 'crt|pem'  
open("/etc/ssl/certs/ca-certificates.crt",  
0_RDONLY) = 5  
$ grep BEGIN \  
    /etc/ssl/certs/ca-certificates.crt | \  
    wc -l  
169
```

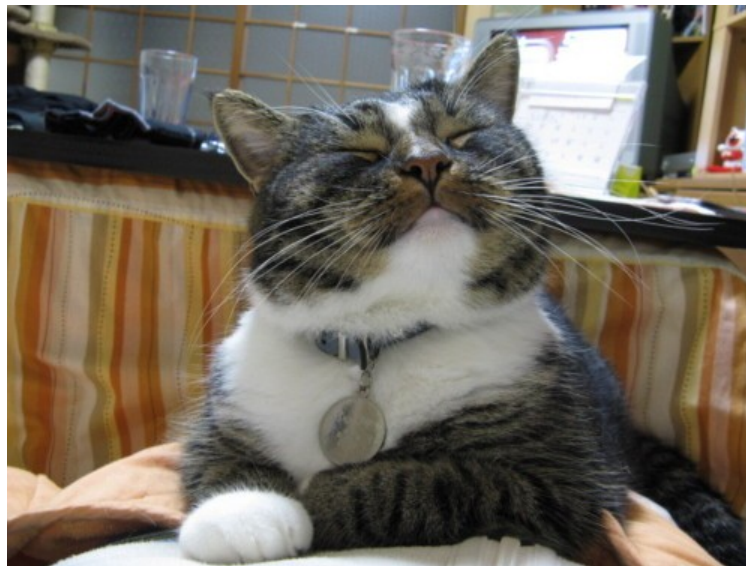
Use “Certificate Pinning”

```
--- https-auth-curl.py 2015-02-01 06:39:07.790892077 -0800
+++ https-auth-curl-ca.py 2015-02-01 22:49:14.091521536 -0800
@@ -2,6 +2,7 @@
import pycurl

curl = pycurl.Curl()
+curl.setopt(pycurl.CAINFO, '/home/kees/osu-devops/ca-bundle.crt')
curl.setopt(pycurl.USERPWD, 'class:omgworstpassword')
curl.setopt(pycurl.URL, "https://outflux.net/osu/devops/secret/data.txt")
```


Test finally notices MitM

- Tiny certificate chain (only the 2 needed certificates)
- Changing host names is left as an additional exercise. Hint: add redirected IP addresses in `/etc/hosts` for the target server



Questions?

<https://outflux.net/slides/2015/osu-devops.pdf>

kees@{outflux.net,debian.org,ubuntu.com}
keescook@{google.com,chromium.org}