API-Induced SSRF

How Apple Pay Scattered Vulnerabilities Across the Web
About me

- Math degree
- Web developer, ~5 years
- Bounties
- At PKC ~1 year, web dev and code audits for clients - pkc.io
Overview

● Definitions
● Demo some mistakes
  ○ Apple Pay
  ○ Twilio
  ○ Others
● How not to be like Apple
Typical Class Breaks

See Schneier’s blog post

Weak Code
(e.g. Heartbleed)

Vulnerable Deployment

Vulnerable Deployment

...
Inductive weakness:

A design flaw that encourages multiple parties to write vulnerable code with a similar exploit pattern across differing software stacks.
SSRF Refresher
Payload with http://169.254.169.254/foo
Payload with http://169.254.169.254/foo
Definitions: SSRF

GET /foo

Payload with http://169.254.169.254/foo

GET /foo

169.254.169.254

sensitive data

sensitive data
If you can relay requests through a GCP or AWS box...

```bash
$ curl -s http://169.254.169.254/computeMetadata/v1beta1/instance/service-accounts/default/token | jq
{
  "access_token": "ya29.c",
  "expires_in": 3511,
  "token_type": "Bearer"
}
```
Easy things to do with SSRF

- AWS, GCP have a gooey center
  - People have already criticized AWS/GCP for this
- `file:///` urls
- Reflected XSS
  - Technically not SSRF
SSRF: Hard mode

- Cross-protocol stuff
  - SMTP through `gopher://` URLs
  - HTTP->memcached->RCE
    - See *A New Era of SSRF*
  - ???
Apple Pay Web
Inductive SSRF
Apple Pay: 3 forms

In-store

In-app

Web

Buy with Apple Pay
these are unaffected
The intended flow

- Safari generates a validationURL
  (https://apple-pay-gateway-*.apple.com)
The intended flow

- Safari generates a validationURL (https://apple-pay-gateway-*.apple.com)
- Your JS sends validationURL to your backend
The intended flow

- Safari generates a validationURL (https://apple-pay-gateway-*.apple.com)
- Your JS sends validationURL to your backend
- Your backend grabs a session from validationURL and forwards it to the client
GET /foo

validationURL
https://169.254.169.254/foo

sensitive data

169.254.169.254
Demos
appr-wrapper

- Under GoogleChromeLabs on github
- Written, deployed by an @google.com account
- A sort of polyfill between Apple Pay and the PaymentRequest API
- A test deployment, so low severity target
.webkit.org

- Maintained by Apple
- Another demo, but on a higher-severity target

Let’s see how this works in a live demo. If you are viewing this post on a device capable of Apple Pay, you should see an Apple Pay button below. Feel free to click it! Don’t worry, no matter what you do in the payment sheet, your card won’t be charged anything.

Genuine Squirrelfish
$20.00 + shipping

Buy with Apple Pay
Apple’s response

Whitelist Apple Pay IP Addresses for Merchant Validation

To enable merchant validation and receive a session object, your server must allow access over HTTPS (TCP over port 443) to the Apple Pay IP addresses and domains provided in Listing 1.

Just added this

Important

Use a strict whitelist for the merchant validation URLs provided by Apple, in Listing 1. Do not allow your server to access any other URLs for merchant validation.
Disclosure timeline

- Feb 11, Initial email to Apple
- March 26, Apple updated docs
- May 14, Apple concluded investigation. I replied with follow-up questions.
- ... Then Apple ghosted for 2 months :(
One mitigation...

Remove apple pay endpoint to remove security vulnerability #72

Merged

merged 1 commit into develop from [redacted] 3 hours ago

Conversation 0  Commits 1  Checks 0  Files changed 1

Changes from all commits  File filter...  Jump to...

Remove apple pay endpoint to remove security vulnerability

develop (#72)

committed 3 hours ago
General mitigations

Apple Pay

- Check `validationURL` against Apple’s list
- Stripe and Braintree handle this flow, so you’re safe if you use them
General mitigations

SSRF in general
● Whitelist egress traffic
● Protect your metadata like Netflix: Detecting Credential Compromise in AWS
● Be mindful of local, unauthenticated stuff on servers
Ineffective mitigations

Do not:

- Use a regex to validate the domain
  - Sometimes people try a regex like `https?://.*.apple.com/.*`
  - But that matches: `http://localhost/?.*apple.com/...`
- Rely on HTTPS to prevent cross-protocol attacks
  - See slide 16 of *A New Era of SSRF*
Webhooks
Inbound Settings

To receive inbound messages on the phone numbers associated with your Messaging Service, you can configure the following settings:

- **PROCESS INBOUND MESSAGES**: Check this box if you want Twilio to make a synchronous HTTP request to the specified URL when it receives an inbound message.

  - **REQUEST URL**: `https://jmaddux.com/sms`
  - **HTTP POST**

- **FALLBACK URL**: Leave this field empty or enter an alternative URL for fallback processing.

  - **HTTP POST**

After configuring the settings, click **Save** to apply the changes.
Previous webhook exploits

Payload would go here

- gopher://localhost:11211/...
Webhook sender

Most attack this

Listener

I’m after these

Listener

Listener

...
How Twilio Authenticates Webhooks

- HMAC and hope the listener checks it
- Lots of webhooks do this, Twilio’s not unique
The problem

- Who failed to check the HMAC?
  - 23 out of 31 open-source projects
The problem

- Who failed to check the HMAC?
  - 23 out of 31 open-source projects
  - Most of Twilio’s example code
The problem

- Who failed to check the HMAC?
  - 23 out of 31 open-source projects
  - Most of Twilio’s example code

- Contributing factors
  - Bad documentation
  - The easiest receiver implementation is a vulnerability
Demo: Webhooks
Twilio Example Code

- Examples themselves not deployed publicly
- But, did find vulns where it was copied/pasted
Disclosure timeline

- Feb 17, Initial email to Twilio
- March 6, Twilio updated some of the docs
  - Rejected all architectural changes due to “unforeseen issues”
What about nexmo?

Validate the signature on incoming messages

In order to verify the origin of incoming webhooks to your SMS endpoint, you can enable message signing for incoming messages - contact support@nexmo.com to request incoming messages be accompanied by a signature. With this setting enabled, the webhooks for both incoming SMS and delivery receipts will include a \textit{sig} parameter. Use the other parameters in the request with your signature secret to generate the signature and compare it to the signature that was sent. If the two match, the request is valid.

Contact support to enable message signing on your account:
support@nexmo.com
What about nexmo?

Validate the signature on incoming messages

Contact support to enable message signing

webhooks for both incoming SMS and delivery receipts will include a **sig** parameter. Use the other parameters in the request with your signature secret. Verify the signature and compare it to the signature that was sent. If the two match, the message was valid.

Contact support to enable message signing on your account:
support@nexmo.com
Gitlab webhooks: the happy path

{  
  "object_kind": "push",  
  "commits": [{  
    "message": "Initial commit of foo project",  
    "url": "https://...",  
    ...  
  }],  
  "repository": {  
    "url": "git@your.git.url/something.git", ...  
  }, ...  
}
What did I do?

- Found a server that was receiving gitlab webhooks
  - On the open internet
  - Was the trigger of build pipelines for multiple tenants...
Gitlab webhooks: what I did

```json
{
    "object_kind": "push",
    "commits": [{
        "message": "Initial commit of foo project",
        "url": "https://...",
        ...
    }],
    "repository": {
        "url": "git@your.git.url/something.git",
        ...
    }
}
```

Put the tenant’s gitlab url here
Gitlab webhooks: what I did

{
  "object_kind": "push",
  "commits": [
    {
      "message": "Click here to do something! :D",
      "url": "javascript:alert('XSS on: ' + window.origin);",
      ...
    }
  ],
  "repository": {
    "url": "git@your.git.url/something.git",
    ...
  }
}

What are some better ways to send webhooks?

- For crypto nerds: authenticated cipher
  - E.g. AES-GCM
  - Still symmetrical like an HMAC
  - Forces webhook consumers to decrypt, so they’ll accidentally verify the GCM tag you send them
What are some better ways to send webhooks?

- More practical: only send high-entropy, cryptographically random event IDs
  - Webhook consumer has to fetch `/items/?id=<id>` with their API token
  - Plaid does roughly this
What are some better ways to send webhooks?

- For existing webhooks: test & warn
  - During registration, do 2 test requests:
    - 1 valid MAC
    - 1 invalid MAC
  - Warn if they get the same response code
What else?
Salesforce Objects vs Dynamodb

Both:
- NoSQL-like object storage
- REST APIs with custom SQL-like queries
Salesforce SOQL

`/?q=SELECT+id+from+Foo+WHERE+name+LIKE+'...'`
Prevent SOQL Injection in Your Code

Learning Objectives

After completing this unit, you'll be able to:

- Learn the different patterns of SOQL injection prevention.
- Prevent SOQL Injection using string.escapeSingleQuotes().
- List the cases where the use of string.escapeSingleQuotes isn’t sufficient.

Source
POST / HTTP/1.1
{
    "TableName": "ProductCatalog",
    "KeyConditionExpression": "Price <= :p",
    "ExpressionAttributeValues": {
        "p": {"N": "500"}
    }
}
Closing Thoughts
From Apple after two months of silence

“Developers are responsible for implementing whatever security and networking best practices make the most sense for their environment.”
“If you’ve built a chaos factory, you can’t dodge responsibility for the chaos.”

Tim Cook, Apple CEO
Financial

- Low-hanging bounty fruit
- Embarrassment
- High-interest tech debt
Designing defensive APIs

- Audit your example code
- Be careful about passing around URLs
- If “Do this or you’re vulnerable!” is in your documentation, try to make the warning unnecessary
Takeaways

● You can find a lot of vulnerabilities by looking at an API, finding a flaw, and seeing who integrates with it.

● We need to place more scrutiny on security weaknesses that induce others to write vulnerable code.

● While there has been a lot of recent work on SSRF, the software development world has a long way to go in defensibly coding around URLs.
Acknowledgments

- Jonathan Ming at PKC - asked the initial questions about Apple Pay
- Arte Ebrahimi at PKC - pointed me to the Nexmo stuff
- Ken Kantzer at PKC - helped with the presentation
- Andrew Crocker at EFF - legal assistance
Thank you!

www.pkci.io