

Security Evaluation on Amazon Web Services' REST API Authentication Protocol Signature Version 4

Khanh Hoang Huynh
Jason Kerssens

Supervisors:
Alex Stavroulakis (KPMG)
Aristide Bouix (KPMG)



UNIVERSITY OF AMSTERDAM

Introduction

- AWS
 - As of 2018, AWS has a dominant share of 47.8% in the cloud service market¹ (IaaS, PaaS)
- Signature Version 4
 - Protocol used for authentication of HTTP API requests
 - Ensures data integrity, verification of the requesting user, and protection against reuse of signed requests
- Other protocols (different functionalities) do not provide end-to-end integrity
 - OAuth 1.0/2.0, SSL/TLS and HTTP Authentication

¹<https://www.ciodive.com/news/iaas-Azure-AWS-Google-Cloud-Alibaba/559716/>

Research question

Does the Signature Version 4 protocol, used when sending a request to AWS REST API endpoints, provide data integrity, verification of the requesting user, and protection against reuse of signed requests?

- How does Signatures Version 4 Protocol ensure data integrity, verification of the requesting user, and protection against reuse of signed requests?
- What kind of attacks are able to undermine data integrity, verification of the requesting user, or protection against reuse of signed requests?

Signature Version 4

- Signing key is derived from the secret access key
- HMAC-SHA1 or HMAC-SHA256
- The signature created is a string in hexadecimal and has a length of 64

1. Creating the canonical request

```
HTTP method + '\n'  
Canonical URI + '\n'  
Canonical Query String + '\n'  
Canonical Headers + '\n'  
Signed Headers + '\n'  
Hashed Payload
```

2. Creating the string to sign

```
Algorithm + '\n'  
Request Date Time + '\n'  
Credential Scope + '\n'  
Hashed Canonical Request
```

3. Creating the signature

Deriving the signing key

```
kSecret = secret access key  
kDate = HMAC("AWS4" + kSecret, Date)  
kRegion = HMAC(kDate, Region)  
kService = HMAC(kRegion, Service)  
SigningKey = HMAC(kService, "aws4_request")
```

```
Signature = HexEncode(HMAC(SigningKey, StringToSign))
```

4. Adding the signature to the request

```
Either added under the Authorization header  
or added in the query string as 'X-Amz-Signature'
```

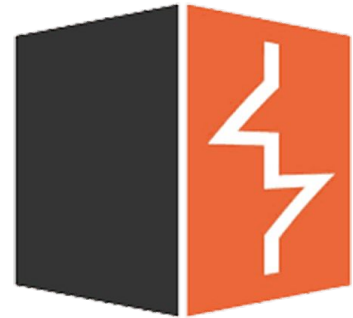
Figure 1: Signature Version 4 signing procedure

Experiments

- Signature Version 4 makes use of HMAC-SHA
 - Attacks on HMAC-SHA are not feasible
- Replay Attack, Modifying Request, HTTP smuggling, and Timing Attacks
- As we look at Signature Version 4, we ignore SSL/TLS
- Attacks were first performed on our local server
- Attacks were then performed on AWS IAM and S3 services

Used Technologies

- Python
- Escher
- Burp



Python Logo source: <https://www.python.org/>

Emarsys Logo (Escher Creator) source: <https://www.emarsys.com/>

Burp Suite logo source: <https://portswigger.net/>

Replay Attack

- Protection against reuse of signed requests
- Intercept request and resend
- For how long?
- What kind of requests?

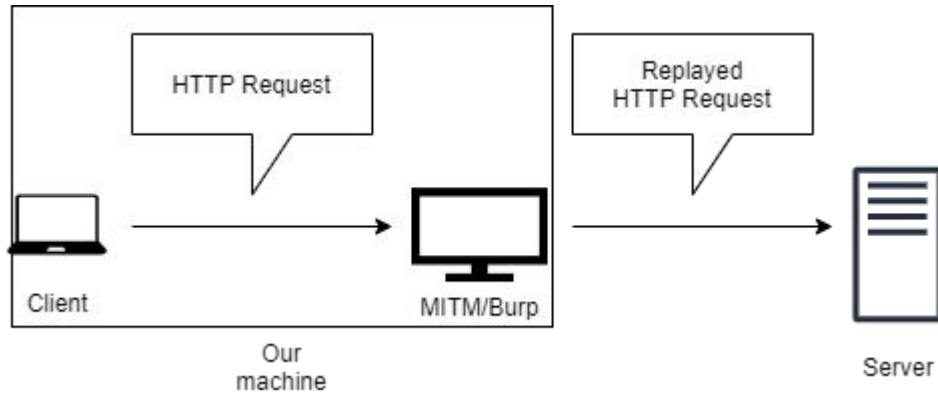


Figure 2: The setup of our replay attack

Modifying the request

- Ensurance of data integrity
- Intercept request, modify it, and send it to intended destination
- What parts of the request can be modified?

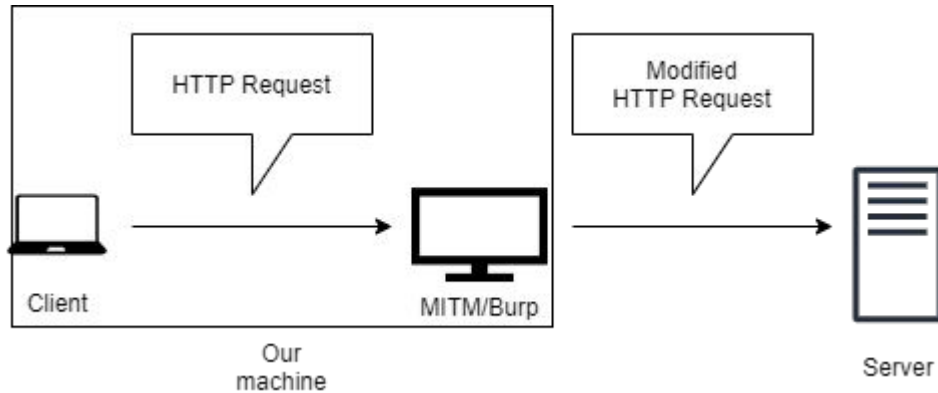


Figure 3: The setup of our modifying the request

HTTP Smuggling

- Verification of the requesting user
- Discrepancy in front-end server and back-end server

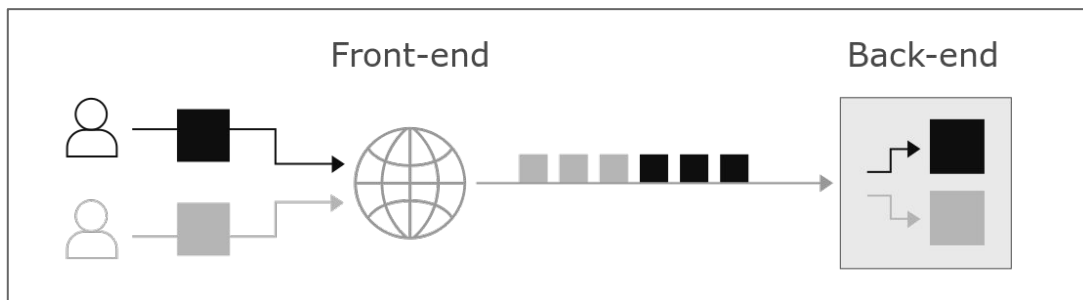


Figure 4: The request flow of modern website architecture. Source: <https://portswigger.net/>

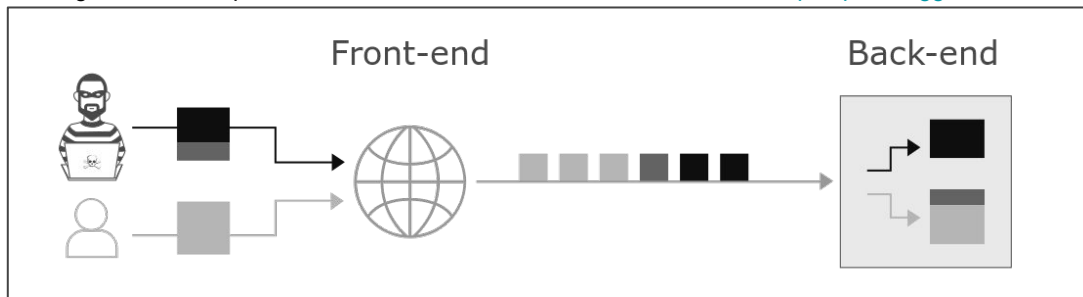


Figure 5: Example of a HTTP request smuggling attack. Source: <https://portswigger.net/>

HTTP Smuggling

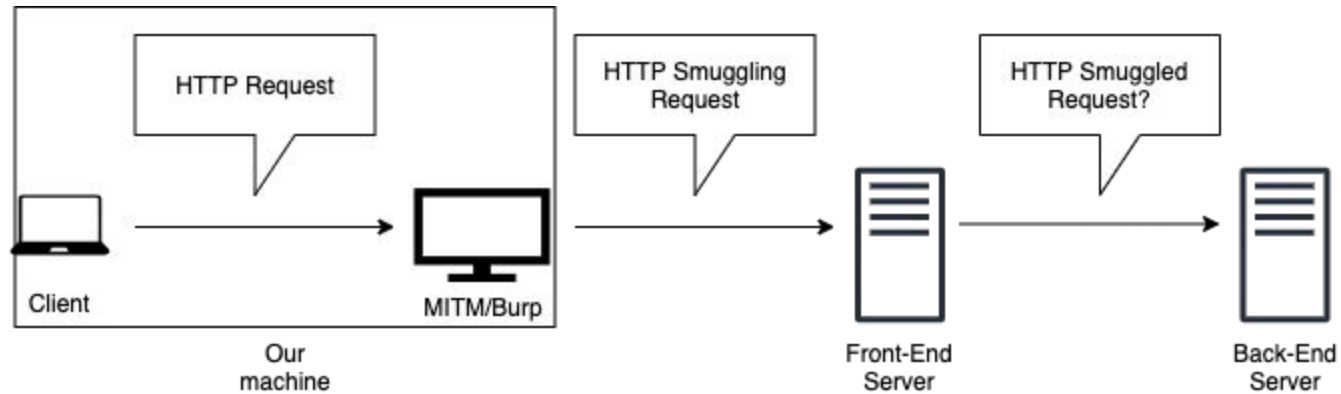
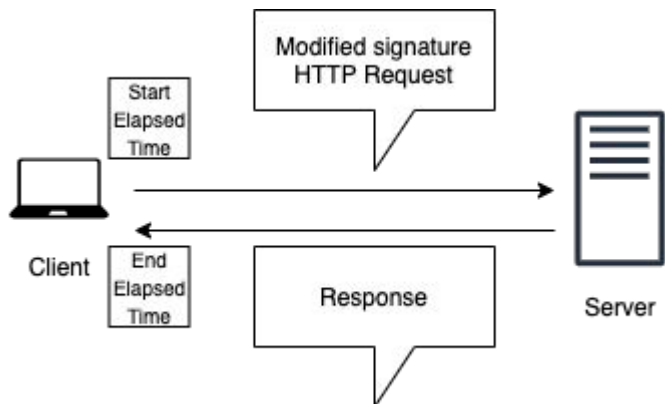


Figure 6: The setup for HTTP smuggling attacks

Probing Timing Attack

- Verification of the requesting user
- Side-channel attack on Signature of HTTP request
- Measure execution time of request and response
- Correlation between execution time and number of valid bits
- Implementation dependent



Example:

'aaaa' != 'aaaa'
'aaaa' != 'aabb'

Figure 7: The setup of our Probing Timing attack

Probing Timing Attack Experiment in detail

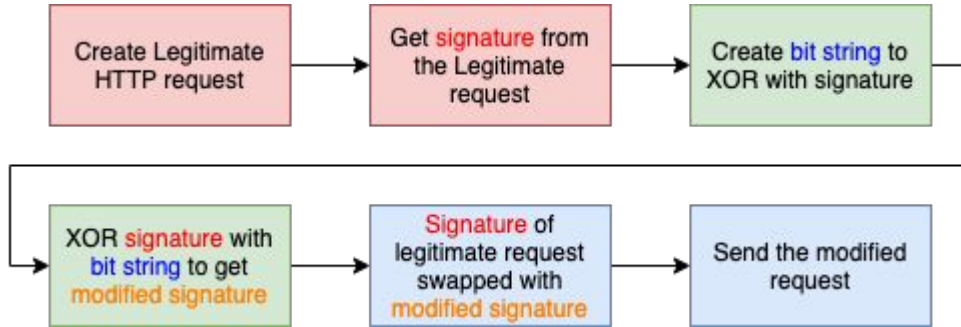


Figure 8: Flow of how we manipulated the signature

```
http://localhost:5000/validate_request?foo=bar&abc=cba&X-EMS-Algorithm=EMS-HMAC-SHA256&X-EMS-Credentials=EXAMPLEAPIKEY%2F20200202%2Fcredential%2Fscope&X-EMS-Date=20200202T143407Z&X-EMS-Expires=1209600&X-EMS-SignedHeaders=host&X-EMS-Signature=e5990c4ea65b5e382f4ce5356b03ca5bf5df4c07be06f84224b7ac4d96dfeda2
```

$1110\dots0010 \oplus 0000\dots0001 = 1110\dots0011$

Figure 9: An example of changing one bit of the correct signature

```
http://localhost:5000/validate_request?foo=bar&abc=cba&X-EMS-Algorithm=EMS-HMAC-SHA256&X-EMS-Credentials=EXAMPLEAPIKEY%2F20200202%2Fcredential%2Fscope&X-EMS-Date=20200202T143407Z&X-EMS-Expires=1209600&X-EMS-SignedHeaders=host&X-EMS-Signature=e5990c4ea65b5e382f4ce5356b03ca5bf5df4c07be06f84224b7ac4d96dfeda3
```



Results (replay attack)

- Replaying of requests possible
- Default valid for 15 mins for IAM
- X-AMZ-Expires option for S3
- Prevented by SSL/TLS

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

73 x 74 x 75 x 76 x 77 x 78 x 79 x 80 x 81 x 82 x 83 x 84 x 85 x 86 x 87 x 88 x 89 x 90 x 91 x 92 x 93 x 94 x 95 x ...

Send Cancel < >

Target: <https://iam.amazonaws.com>  

Request

Raw Params Headers Hex

```
GET
/?Action=ListUsers&Version=2010-05-08&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAW42NG6RQO6LML5RB%2F20200202%2Fus-east-1%2Fiam%2Faws4_request&X-Amz-Date=20200202T182835Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-Signature=37f1e4e98bb59555c43e89ff806c0c6412705ca427f6efa63cb1549b94d2d80 HTTP/1.1
Host: iam.amazonaws.com
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
```

Response

Raw Headers Hex XML

```
HTTP/1.1 200 OK
x-amzn-RequestId: 80f7a4e6-e763-469e-8a1f-e3c7a6efa470
Content-Type: text/xml
Content-Length: 545
Date: Sun, 02 Feb 2020 18:29:16 GMT
Connection: close

<ListUsersResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
  <ListUsersResult>
    <IsTruncated>false</IsTruncated>
    <Users>
      <member>
        <Path></Path>
        <UserName>Test</UserName>
        <Arn>arn:aws:iam:474218951776:user/Test</Arn>
        <UserId>AIDAW42NG6RQD6SCFVI43</UserId>
        <CreateDate>2020-01-16T22:09:19Z</CreateDate>
      </member>
    </Users>
  </ListUsersResult>
  <ResponseMetadata>
    <RequestId>80f7a4e6-e763-469e-8a1f-e3c7a6efa470</RequestId>
  </ResponseMetadata>
</ListUsersResponse>
```

Figure 10: HTTP Request and response

Request

Raw Params Headers Hex

```
GET
/?Action=ListUsers&Version=2010-05-08&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credent
ial=AKIAW42NG6RQO6LML5RB%2F20200202%2Fus-east-1%2Fiam%2Faws4_request&X-Amz-Date=2020
0202T182835Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-Signature=37f1e4e98b
bb59555c43e89ff806c0c6412705ca427f6efa63cb1549b94d2d80 HTTP/1.1
Host: iam.amazonaws.com
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
```

Response

Raw Headers Hex XML

```
HTTP/1.1 200 OK
x-amzn-RequestId: 3efd0d32-f475-48ef-b813-75a953a7447d
Content-Type: text/xml
Content-Length: 545
Date: Sun, 02 Feb 2020 18:29:52 GMT
Connection: close

<ListUsersResponse xmlns="https://iam.amazonaws.com/doc/2010-05-08/">
  <ListUsersResult>
    <IsTruncated>false</IsTruncated>
    <Users>
      <member>
        <Path>/</Path>
        <UserName>Test</UserName>
        <Arn>arn:aws:iam::474218951776:user/Test</Arn>
        <UserId>AIDAW42NG6RQD6SCFVI43</UserId>
        <CreateDate>2020-01-16T22:09:19Z</CreateDate>
      </member>
    </Users>
  </ListUsersResult>
  <ResponseMetadata>
    <RequestId>3efd0d32-f475-48ef-b813-75a953a7447d</RequestId>
  </ResponseMetadata>
</ListUsersResponse>
```

Figure 11: Replayed a HTTP Request and response

Results (modifying requests)

- Signed parts cannot be changed
- S3 unsigned payload option
- Prevented by SSL/TLS

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

📄 Request to https://s3.amazonaws.com:443 [52.216.224.243]

Forward Drop Intercept is on Action Comment this item 🌈 ?

Raw Params Headers Hex

```
PUT /TestBucket2/Test3 HTTP/1.1
Host: s3.amazonaws.com
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
x-amz-content-sha256: UNSIGNED-PAYLOAD
x-amz-date: 20200202T184947Z
x-amz-expires: 600
Authorization: AWS4-HMAC-SHA256 Credential=AKIAW42NG6RQ06LML5RB/20200202/us-east-1/s3/aws4_request, SignedHeaders=host;x-amz-content-sha256;x-amz-date;x-amz-expires,
Signature=6dd99ec4760e0567663e573b7b33663e055eb9d0303300db85f854d06aed4333
Content-Length: 160
Content-Type: multipart/form-data; boundary=dcce6570a57a1380e593d1a4b4d53818

--dcce6570a57a1380e593d1a4b4d53818
Content-Disposition: form-data; name="file"; filename="test.txt"
OS3 is the best!
--dcce6570a57a1380e593d1a4b4d53818--
```

Figure 12: HTTP request payload to be modified

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

✍️ Request to https://s3.amazonaws.com:443 [52.216.224.243]

Forward Drop Intercept is on Action [Comment this item](#) ?

Raw Params Headers Hex

```
PUT /TestBucket2/Test3 HTTP/1.1
Host: s3.amazonaws.com
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
x-amz-content-sha256: UNSIGNED-PAYLOAD
x-amz-date: 20200202T184947Z
x-amz-expired: 600
Authorization: AWS4-HMAC-SHA256 Credential=AKIAW42NG6RQ06LML5RB/20200202/us-east-1/s3/aws4_request, SignedHeaders=host;x-amz-content-sha256;x-amz-date;x-amz-expired,
Signature=6dd99ec4760e0567663e573b7b33663e055eb9d0303300db85f854d06aed4333
Content-Length: 160
Content-Type: multipart/form-data; boundary=dcce6570a57a1380e593d1a4b4d53818


--dcce6570a57a1380e593d1a4b4d53818
Content-Disposition: form-data; name="file"; filename="test.txt"
OS3 is so-so!
--dcce6570a57a1380e593d1a4b4d53818--
```

Figure 13: HTTP request payload changed and sent

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

73 x 74 x 75 x 76 x 77 x 78 x 79 x 80 x 81 x 82 x 83 x 84 x 100 x 101 x 102 x ...

Send Cancel < ▾ > ▾

Target: <https://s3.amazonaws.com> 

Request

Raw Params Headers Hex

```
GET
/TestBucket2/Test3?Action=GetObject&Bucket=TestBucket2&Key=Test2&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAW42NG6RQO6LML5RB%2F20200126%2Fus-east-1%2Ffs3%2Faws4_request&X-Amz-Date=20200126T185211Z&X-Amz-Expires=604800&X-Amz-SignedHeaders=host&X-Amz-Signature=9dcab6851dc1c5b219a51a4ca7a488a015703ac15757d73545a3c1a896c99bf9
HTTP/1.1
Host: s3.amazonaws.com
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
```

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
x-amz-id-2:
/CkWvV7gxNGA+/YYTJ/FjImqAmJa3nKwa5BBFxp4rJnqDxn7IFqkPyRZHTW12mpXaPymjFxdH34=
x-amz-request-id: F0F565918D9C1DFD
Date: Sun, 02 Feb 2020 18:51:36 GMT
Last-Modified: Sun, 02 Feb 2020 18:50:53 GMT
ETag: "2259fea9b644875b545b72816c02581a"
Accept-Ranges: bytes
Content-Type: multipart/form-data; boundary=dcce6570a57a1380e593d1a4b4d53818
Content-Length: 157
Server: AmazonS3
Connection: close

--dcce6570a57a1380e593d1a4b4d53818
Content-Disposition: form-data; name="file"; filename="test.txt"

OS3 is so-so!
--dcce6570a57a1380e593d1a4b4d53818--
```

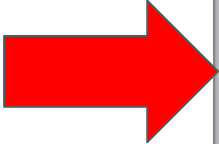
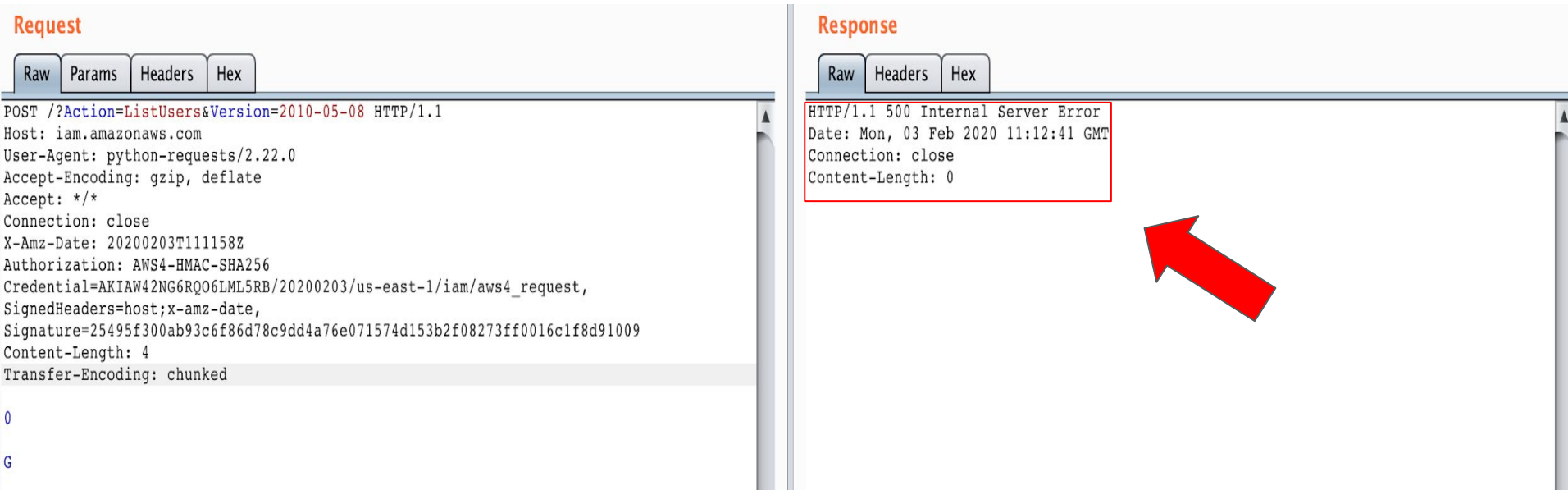


Figure 14: Successfully modified HTTP request and uploaded to AWS

Results (HTTP Smuggling)

- Not successful, as AWS responds with HTTP Status Code 500



The image shows a network traffic analysis tool interface with two panels: Request and Response.

Request Panel: The 'Raw' tab is selected. The request is a POST to `/?Action=ListUsers&Version=2010-05-08` on `iam.amazonaws.com`. The headers include `User-Agent: python-requests/2.22.0`, `Accept-Encoding: gzip, deflate`, `Accept: */*`, `Connection: close`, `X-Amz-Date: 20200203T111158Z`, `Authorization: AWS4-HMAC-SHA256`, `Credential=AKIAW42NG6RQ06LML5RB/20200203/us-east-1/iam/aws4_request,`, `SignedHeaders=host;x-amz-date,`, `Signature=25495f300ab93c6f86d78c9dd4a76e071574d153b2f08273ff0016c1f8d91009`, `Content-Length: 4`, and `Transfer-Encoding: chunked`.

Response Panel: The 'Raw' tab is selected. The response is `HTTP/1.1 500 Internal Server Error`. The headers include `Date: Mon, 03 Feb 2020 11:12:41 GMT`, `Connection: close`, and `Content-Length: 0`. A red box highlights the status code and message, and a red arrow points to it from the right.

Figure 15: Result of executing the detecting if http smuggling is possible

Results (Timing attack)

- Escher
- Correlation found? (between execution time and number of correct bits in signature)

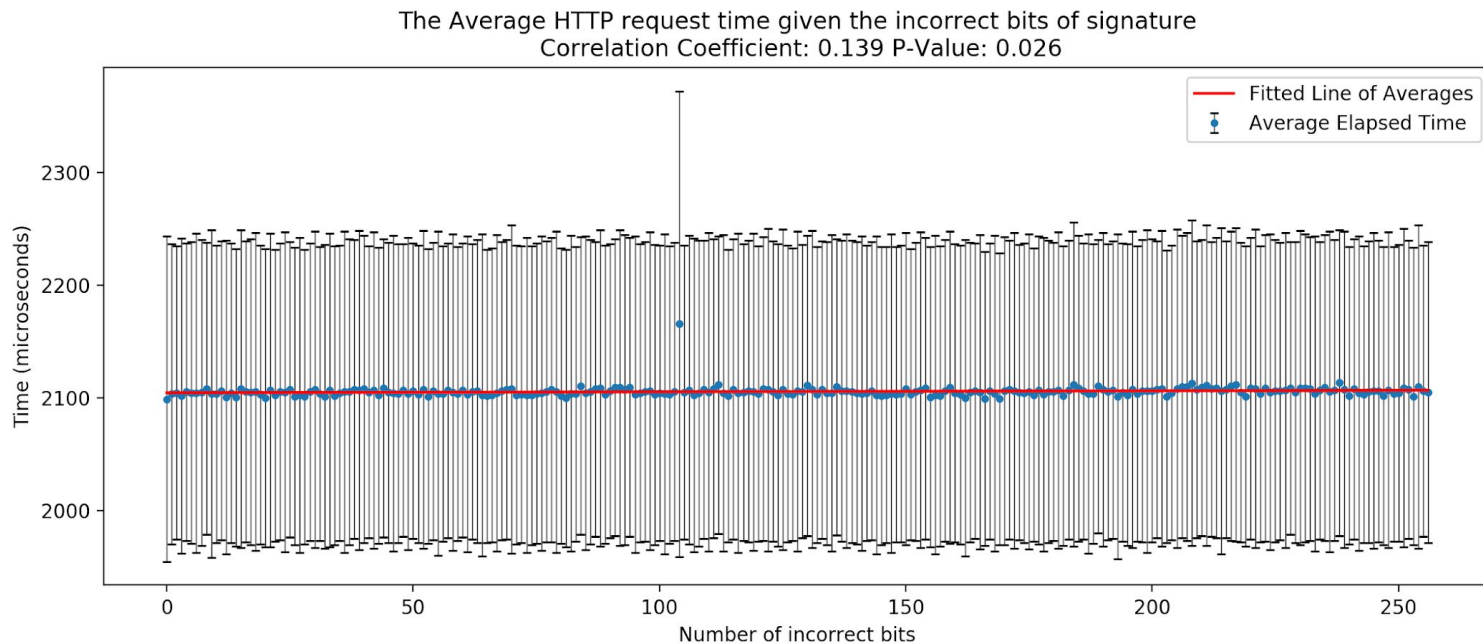


Figure 16: The results of seeing if a timing attack would be effective

Results (Timing attack)

The Average HTTP request time given the incorrect bits of signature
Correlation Coefficient: 0.139 P-Value: 0.026

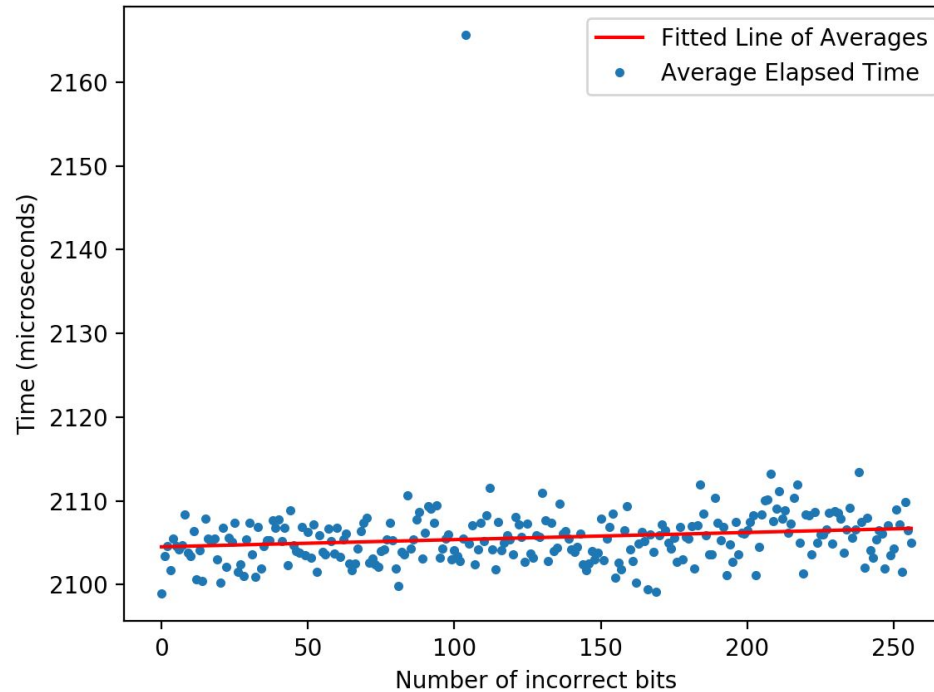


Figure 17: Figure 15, but without the standard deviation plotted

Conclusion

How does Signature Version 4 ensure protection?

- Data integrity: Signature
- User verification: API KEY ID, and Secret Access Key
- Reuse of signed portions: Expiration of request

What kind of attacks are possible?

- Replay attack: reuse of signed portions is possible for a limited time
- Modifying requests: signed portions of requests cannot be modified, unsigned portions can be modified
- HTTP Smuggling: not successful
- Timing attack: correlation found locally

Conclusion

Does the Signature Version 4 protocol, used when sending a request to AWS REST API endpoints, provide data integrity, verification of the requesting user, and protection against reuse of signed requests?

- Provides data integrity of signed portions
- Verifies that signed parts were indeed signed by user
- Does not fully provide protection of reuse of signed portions

Future work

- Other services
- Timing attack on AWS servers
- Inspect the SSL/TLS from AWS API endpoint

Conclusion

Does the Signature Version 4 protocol, used when sending a request to AWS REST API endpoints, provide data integrity, verification of the requesting user, and protection against reuse of signed requests?

- Provides data integrity of signed portions
- Verifies that signed parts were indeed signed by user
- Does not fully provide protection of reuse of signed portions