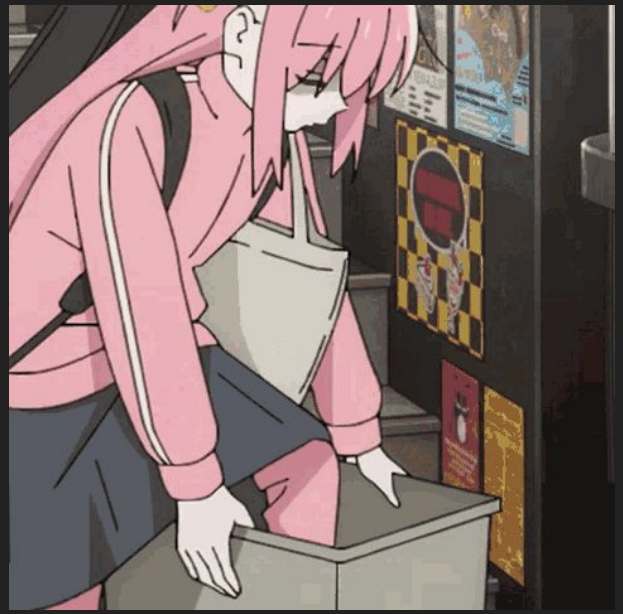
Writing Your First Burp Extension

Marcus Chan @ benkyou

DISCLAIMER!!!

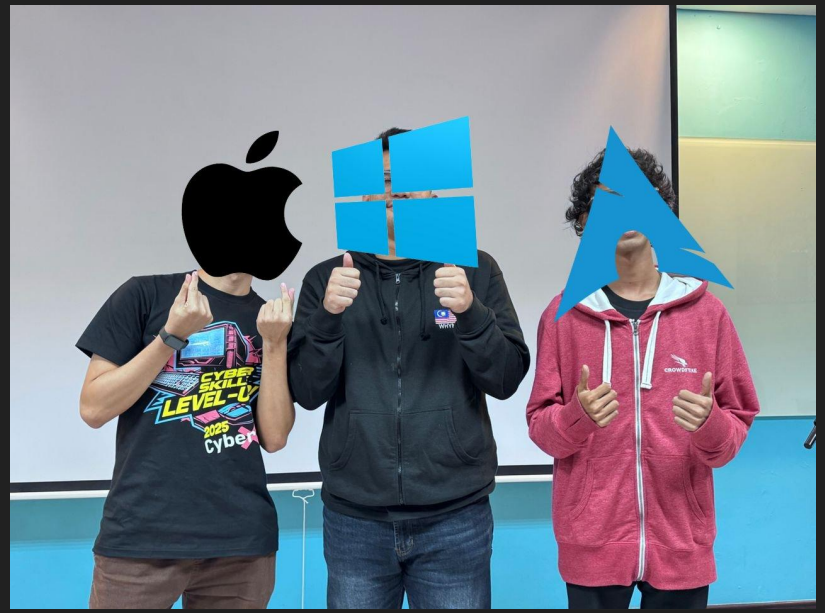
- I am not a professional developer and some of the code here may be *downright atrocious*
- However… the code works good enough for our use case :D
- ~~I may have no idea what I am talking about~~ 🤷

We won’t really be hacking today…



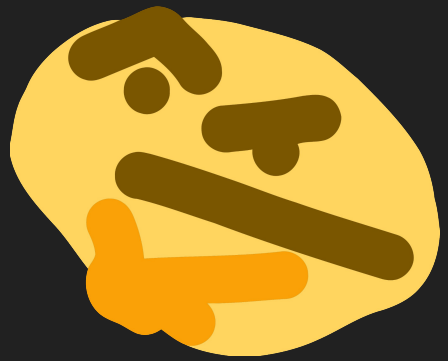
% whoami

- Marcus Chan
- Computer Science @ USM (“previously”)
- I play CTFs at RE:UN10N :D



Why should I even care?

- Good to know how your tools work and how to customize them to your workflow
- Some assessments may not be possible
  - Business logic that gets in your way of testing



Example Scenario (Signed requests)

- Commonly used for payment-related API requests
- Application requires hash of your request body to be set in a header
- This means if we modify any parts of the body, our request will get rejected if we don’t calculate a new hash
- At this point, you’re stuck 🗿

Ref:https://docs.developer.paynet.my/docs/duitNow-online-banking-wallets/integration/security-&-encryption/message-signature/JSON-web-signature

Couple of Solutions

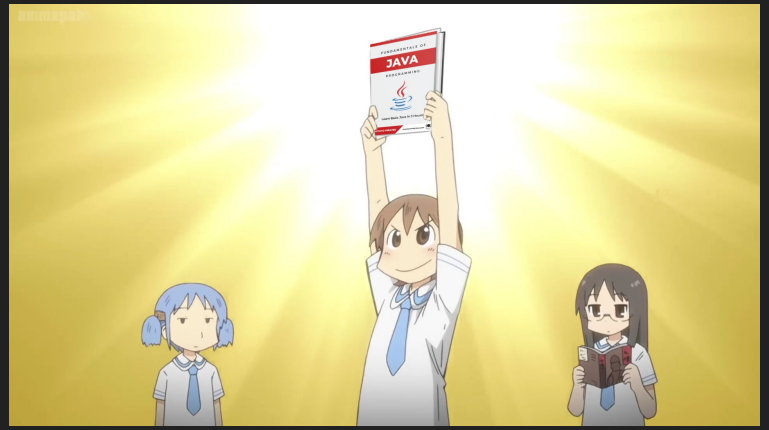
1. Manually generate the hash for each new request 💀
2. Write a python script to do calculation then using `requests`

PROBLEMS:

1. Tedious, and it gets annoying very fast
2. Both methods are very slow…
3. You can’t use any of your other tools in Burp, sqlmap, brute forcing, etc.

Introducing Burp Montoya API

- The `HttpHandler` interface allows us to manipulate all HTTP requests to Burp (`burp.api.montoya.http.handler`)
- This allows us to proxy requests from other tools to Burp to calculate the appropriate headers
- Very simple, we only need to implement 2 methods
- Can write in Python (Jython), R (Jruby), Java



Burp Suite Community Edition v2025.7.4 - Temporary Project

Dashboard | Target | Proxy | Intruder | Repeater | Collaborator | Sequencer | Decoder | Comparer | Logger | Organizer

Extensions | Learn

1

Installed | BApp Store | APIs | BChecks | Bambda library | Extensions settings

Burp Extender APIs

You can use the Burp Extender APIs to create your own extensions to customize Burp's behavior.

2

Download starter project

BurpExtension

Select BurpExtension

EnhancedCapability
MontoyaApi
- ai
- bambda
- burpsuite
- collaborator
- comparer
- core
- decoder
- extension
- http
- internal
- intruder
- logger
- logging
- organizer
- persistence
- project
- proxy
- repeater
- scanner
- scope
- sitemap
- ui
- utilities
- websocket

```
/*
 * Copyright (c) 2022-2023. PortSwigger Ltd. All rights reserved.
 *
 * This code may be used to extend the functionality of Burp Suite Community Edition
 * and Burp Suite Professional, provided that this usage does not violate the
 * license terms for those products.
 */

package burp.api.montoya;

import java.util.Set;

import static java.util.Collections.emptySet;

/**
 * All extensions must implement this interface.
 * <p>
 * Implementations must be declared public, and must provide a default (public, no-arg
 */
public interface BurpExtension
{
    /**
     * Invoked when the extension is loaded. Any registered handlers will only be enab
     *
     * @param api The API implementation to access the functionality of Burp Suite.
     */
    void initialize(MontoyaApi api);
```

Search | 0 highlights

Download interface files | Download Javadoc files

Then you can import the project to your favourite IDE/editor

Our HelloWorld!

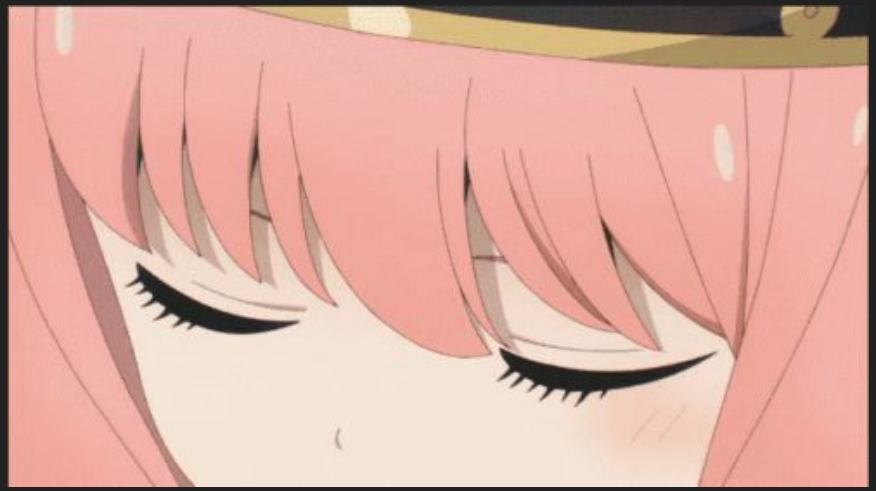
- Simple signing algorithm
- Link to web app source code here

Validation steps:

1. Check if Signature header is set
2. Get request body then hash with SHA256
3. Compare Signature with calculated hash
4. Forward request if match, else reject

https://github.com/benkyousec/writing-your-first-burp-extension/tree/main/helloworldv1-api

That was easy enough



Now let’s add more complexity :>

HelloWorld v2

- All requests must set the `Timestamp`, `Ref`, and `Signature` custom headers
- `Timestamp` must be within 10 second timeframe when server receives the request.
- Each request must have a unique `Ref` header value. For example, TESTER000000000 to identify the person assigned, and 9 digits for tagging
- Each request must be signed and set its calculated value in the `Signature` header.

https://github.com/benkyousec/writing-your-first-burp-extension/tree/main/helloworldv2-api

Our “improved” signing algorithm

Header:

{"alg":"HS256","typ":"JWT","uri":"/API-ENDPOINT-HERE","iat":"02052024133409"}

Timestamp in ddMMyyyyHHmmss format. Also need to update uri for each endpoint. Header must also be minified.

Payload:

{“id”:”1”,”amount”:”4”,”recipient”:”user@example.com”}

Payload must also be minified

Cont.

SigningInput = Base64(Header).Base64(Payload)

Signature = HS256(SigningInput, SECRET_KEY)

Set `Signature` header : Base64(Header).Base64(Payload).Base64(Signature)

https://github.com/benkyousec/writing-your-first-burp-extension/tree/main/helloworldv2-extension

```go
func (e *Env) GetQuote(c *gin.Context) {
...[SNIP]...
    query := fmt.Sprintf("SELECT text FROM quote WHERE id=%s LIMIT 1", reqId.Id)

    var quote sql.NullString
    err := e.db.QueryRow(query).Scan(&quote)
    if err != nil || !quote.Valid {
        c.JSON(http.StatusOK, gin.H{
            "message": "No quote found",
        })
        return
    }

    c.JSON(http.StatusOK, gin.H{
        "quote": quote.String,
    })
}
```

Some tips from my experience

- Always read the documentation.
- If you use a JWT library it may give you unexpected results when the API uses a very custom algorithm. You should implement it yourself.
- Unless the feature you need for your extension is very simple, I'd suggest writing in Java.
- If you deal with Base64 encoding, double check if it's standard Base64 (uses +, /) or Raw Base64 (uses -, _)

References

- https://docs.developer.paynet.my/docs/duitNow-online-banking-wallets/integration/security-&-encryption/message-signature/JSON-web-signature
- https://portswigger.github.io/burp-extensions-montoya-api/javadoc/burp/api/montoya/MontoyaApi.html