NoSQL injection

إيه هو NoSQL Injection؟

NoSQL Injection هي ثغرة أمنية بتخلي المهاجم يقدر يتدخل في الـ queries اللي التطبيق بيعملها على قاعدة بيانات NoSQL. يعني إيه؟ تخيّل إن التطبيق بتاعك بيبعت طلبات لقاعدة البيانات، لو المدخلات (الـ inputs) مش متأمّنة كويس، المهاجم ممكن يحط كود أو حاجات غريبة تخلي قاعدة البيانات تتصرف بطريقة مش متوقعة.

المهاجم ممكن يعمل إيه لو استغل الثغرة دي؟

- يخترق الحماية، يعني يعدّي الـ authentication من غير ما يكون عنده باسورد صح.
- يسرق بيانات أو يعدّل عليها.
- يعمل denial of service، يعني يخلي التطبيق يقف خالص.
- ينفذ كود على السيرفر نفسه، ودي أخطر حاجة.

NoSQL databases بتخزن البيانات بطريقة مختلفة عن الـ SQL databases التقليدية. الـ SQL بيستخدم جداول مرتبة وعلاقات (relations)، لكن الـ NoSQL بيستخدم هياكل بيانات زي الـ documents أو الـ key-value stores، وممكن يستخدم لغات query مختلفة جدًا. عشان كده، الثغرات بتكون مختلفة شوية.

أنواع NoSQL Injection

هنتكلم عن نوعين رئيسيين لـ NoSQL Injection:

1. Syntax Injection

ده يعني إن المهاجم بيحاول يكسر صيغة الـ query بتاعة قاعدة البيانات بحيث يحط payload (يعني كود أو حاجة يضر بيها). الطريقة دي تشبه الـ SQL Injection، بس بتختلف عشان لغات الـ NoSQL وهياكل البيانات متنوعة جدًا.

إزاي بيكتشفوها؟

- بيبعتو fuzz strings (يعني كلام عشوائي أو حروف خاصة زي `"'` أو `$`) عشان يشوفو لو قاعدة البيانات هتطلّع error أو تتصرف بطريقة غريبة لو المدخلات مش متأمّنة.
- لو عارف نوع قاعدة البيانات (زي MongoDB مثلاً)، بيستخدمو fuzz strings مخصصة للغة الـ API بتاعتها. لو مش عارفين، بيجربو حاجات كتير.
- مثال لـ fuzz string في MongoDB لو في URL parameter:

```
`Foo \xYZ$ {Foo$; }"'
```

ده لازم يتعمل URL-encoded عشان يشتغل في الرابط.

- مثال تاني لو في JSON property:

```
`n$Foo \xYZ\{r;$Foo\}"\"
```

- لو الـ response اتغير، يبقى المدخلات مش متأمّنة.
- ممكن يجربو حروف زي `'` لوحدها، يشوفو لو بتكسر الـ query. لو هرّبت الحرف (زي `'\`) والـ query اشتغلت، يبقى فيه ثغرة.

2. Operator Injection

هنا المهاجم بيستخدم query operators (زي `$` `$regex` , `$in` , `$ne`, `where`) عشان يعبث في الـ query. الـ operators دي بتكون جزء من لغة قاعدة البيانات نفسها.

استغلال NoSQL Injection (هنركز على MongoDB)

اكتشاف وتأكيد Syntax Injection

- **تأكيد السلوك الشرطي**: لو اكتشفت ثغرة، لازم تشوف لو الـ boolean conditions ممكن تتأثر. يعني ابعت طلب بـ condition غلط (زي `x' && 0 && '`) وطلب بـ condition صح (زي `x' && 1 && '`). لو التطبيق تصرف بطريقة مختلفة، يبقى الـ query على السيرفر اتأثر.
- **تخطي الشروط**: لو الـ conditions بتتأثر، حط JavaScript condition دايمًا بتطلع true (زي `'1'=='1'||`) عشان تخطّي منطق الـ query الأصلي. ده ممكن يخلي الـ query ترجّع كل البيانات، حتى اللي المفروض تكون مخفية. بس خد بالك، لو الـ query دي بتستخدم لـ update أو delete، ممكن تخسر بيانات!
- **استخدام null character**: حط null character (`%00` في الـ URL) بعد المدخل. MongoDB ممكن يتجاهل كل الحروف بعد الـ null character، فده بيخلّي الـ query تتعدّى شروط زي `this.released == 1` ، ويطلّع بيانات المفروض تكون مقيّدة.

اكتشاف واستغلال Operator Injection

- قواعد البيانات NoSQL بتستخدم operators زي `$where`, `$ne` , `$in` , `$regex` عشان تحدد الشروط.
- ممكن تحقن الـ operators دي في JSON زي:

```
{username":{"$ne":"invalid"}"} → {"username":"wiener"}
```

- لو بتستخدم URL parameters، ممكن تجرب تحقن operator زي `username[$ne]=invalid` .
- لو الـ URL parameters مش شغالة، غيّر الطلب لـ POST، خلّي الـ Content-Type بتاعه `application/json` ، وحط JSON في الـ body. أداة زي Burp Suite مع الـ Content Type Converter extension بتساعد في ده.
- لو بتختبر JSON request body (زي `{"username":"wiener","password":"peter"}`)، جرب operators. مثال:

```
{"username":{"$ne":"invalid"},"password":"peter"}
```

ده بيسال عن المستخدمين اللي الـ username بتاعهم مش `invalid` .

- لو الـ username والـ password بيقبلو operators، ممكن تعمل authentication bypass. مثال:

```
{username":{"$ne":"invalid"},"password":{"$ne":"invalid"}"}
```

ده هيرجّع أول مستخدم في الـ collection، يعني ممكن تدخل من غير بيانات صحيحة.

- لو عايز تستهدف حساب معين (زي `admin`):

```
{username":{"$in":["admin","administrator","superadmin"]},"password":{"$ne":""}"}
```

ده بيجرب يدخل بحسابات admin مختلفة مع باسورد مش فاضي.

استخراج البيانات باستخدام Operators/JavaScript

- بعض operators زي `where$` أو functions زي `mapReduce()` في MongoDB بتسمح بتنفيذ JavaScript محدود. لو التطبيق بيستخدمهم، قاعدة البيانات ممكن تنفّذ JavaScript محقون.
- ممكن تستخدم JavaScript عشان تستخرج بيانات. مثال: لو الـ query بتستخدم `where$` زي `{"'where":"this.username == 'admin$"}` ، جرب تحقن:

```
' && this.password == 'a' || 'a'=='b
```

ده هيرجّع أول حرف من الباسورد بتاع `admin` .

- ممكن تستخدم دالة `match()` في JavaScript (زي `' &&` `this.password.match(/\d/) || 'a'=='b`) عشان تختبر أنماط زي لو فيه أرقام في الباسورد.
- Lab PRACTITIONER: استغلال NoSQL injection عشان تستخرج بيانات.

تحديد أسماء الحقول

في قواعد بيانات زي MongoDB، البيانات بتكون semi-structured، يعني لازم تعرف أسماء الحقول (زي `username` أو `password`). إزاي؟

- اختبر وجود الحقول بـ conditions. مثال:

```
' && this.password && '!=username=admin'
```

قارن الـ response لـ:

  - حقل موجود زي `username` .
  - حقل متوقع زي `password` .
  - حقل مش موجود زي `foo` .

- ممكن تستخدم dictionary attacks مع wordlists عشان تتوقع أسماء الحقول.

استخراج أسماء الحقول باستخدام Operators

- حتى لو الـ query الأصلية مش بتستخدم JavaScript-enabled operators، ممكن تحقن واحد زي `where$` .
- اختبر الحقن بحيث تضيف operator مع boolean conditions:

```
{"username":"wiener","password":"peter","$where":"0"}
```

و

```
{"username":"wiener","password":"peter","$where":"1"}
```

لو الـ response مختلف، يبقى الـ JavaScript في `where$` بيتحلّل.

- لو الـ JavaScript بيتحلّل، استخدم دالة `()keys` عشان تستخرج أسماء الحقول.

مثال:

```
"where":"Object.keys(this).match('^.{0}a.*')$"
```

ده بيفحص أول حرف من أول اسم حقل.

Lab PRACTITIONER: استغلال NoSQL operator injection عشان تستخرج حقول مجهولة.

استخراج البيانات من غير JavaScript

ممكن تستخدم operators زي `regex$` عشان تستخرج بيانات حرف بحرف:

- اختبر لو `regex$` بيشتغل:

```
{username":"admin","password":{"$regex":"^.*"}"}
```

لو الـ response مختلف عن محاولة باسورد غلط، يبقى فيه ثغرة.

- استخدم `regex$` عشان تفحص حروف: `username":"admin","password":"}` `{{"$regex":"^a*"}`

ده بيشوف لو الباسورد بيبدأ بحرف `a` .

Timing Based Injection

لو الـ errors أو الـ conditions مش بتغيّر الـ response، جرب JavaScript يعمل تأخير زمني:

- حدد وقت تحميل الـ baseline.
- حقن payload يعمل تأخير (زي `{"where":"sleep(5000)$"}` لتأخير 5 ثواني).
- لو الـ response أبطأ، يبقى الحقن نجح.
- ممكن تعمل conditional timing payloads (يعني التأخير يحصل بس لو أول حرف في الباسورد `a`).

نصائح للوقاية

- نظّف وتحقق من المدخلات باستخدام allowlist للحروف المسموح بيها.
- استخدم parameterized queries بدل ما تدمج المدخلات مباشرة في الـ queries.
- عشان تمنع operator injection، حدد allowlist للـ keys المسموح بيها في الـ query objects.
- راجع الـ security documentation بتاعة قاعدة البيانات NoSQL اللي بتستخدمها.

تم بحمد الله

Contact

For questions, feedback, or support:

- X (Formly Twitter): @00xmora
- Linkedin: @00xmora