12 Minutes to Exposure

THE RESULT

12 min
From zero access to confirmed exposure

1
Confirmed high-severity finding

100%
Maintained same standards for proof and validation

Apex UNAUTHENTICATED WEB APP AGENT

THE FOLLOWING ARE APEX LOGS AND COMMANDS FROM A PRODUCTION EXECUTION, ANONYMIZED FOR DEMONSTRATION.

[21:29:22]

A Apex

It starts methodically.

Apex opens with a plan to read the application before touching it. The partner-context engine is loaded in the background and gets called at the end, in step 8.

```
[21:29:22] Let me start by performing thorough reconnaissance on the target. I'll begin by
fetching the main page to understand the application, checking what technologies are in use, and
mapping the attack surface.
```

```
$ curl -sk -D- https://example.acme.com/v20/
-------------------------------------------------------------------------------------------------
HTTP/2 302
location: /v20/Login.aspx?ReturnUrl=%2fv20%2fDefault.aspx
server:   cloudflare
x-aspnet-version: 4.0.30319
x-miniprofiler-ids: ["6ca4d940-9895-4272- ... "]
```

[21:29:50]

A Apex

It fingerprints the application.

Apex reads the page and the client-side JavaScript and builds a working model of what it is dealing with, the way an experienced tester would before touching anything.

```
[21:29:50] This is an ASP.NET web application with a login page, a password reset page, and a
single sign-on option that is hidden by default. The current user is empty, as expected for an
unauthenticated session. I should dig into the JavaScript files and check for paths or endpoints
that reveal more about the application's structure.
```

```
$ curl -sk https://example.acme.com/v20/Login.aspx
curl -sk https://example.acme.com/v20/scripts/page_init.js
-------------------------------------------------------------------------------------------------
ASP.NET 4.0 WebForms  ·  DevExpress controls  ·  jQuery 3.1.1
__VIEWSTATE present  ·  hidden SSO option  ·  currentUser = ''
```

[21:30:26]

A Apex

It maps the surface and forms a hypothesis.

Apex finds the back-end services the page calls at runtime by reading the application's own JavaScript. That gives it a specific, testable hypothesis to work from.

```
[21:30:26] One service requires authentication, but a second lookup service is accessible without
it, and exposes a method whose parameters reference database table and field names directly.
These look like potential SQL injection vectors, so I should test them.
```

```
$ curl -sk 'https://example.acme.com/v20/services/Lookup.asmx?WSDL'
-------------------------------------------------------------------------------------------------
<s:element name="GetLookupResults">
  <s:element name="lookupTable"  type="s:string" />
  <s:element name="lookupField"  type="s:string" />
  <s:element name="lookupOption" type="s:string" />
  <s:element name="term"         type="s:string" />
```

[21:30:58]

A Apex

It tells open doors from locked ones.

Apex sweeps the rest of the application and reads each response for what it actually means. A 403 says the door is closed; it does not say there is something to report.

```
[21:30:58] The sensitive directories and the configuration file are blocked, which is good. The
lookup service accepts requests but returns no data regardless of input, and the injection
attempts do not error out, which could mean the input is being sanitized or there is simply no
data behind it.
```

```
$ for p in Default.aspx Admin/ Services/ web.config Trace.axd \
        mini-profiler-resources/results WebService.asmx; do
  echo "$(curl -sk -o/dev/null -w '%{http_code}'  .../v20/$p) - $p"
done
-------------------------------------------------------------------------------------------------
302 - Default.aspx                      (login redirect, as expected)
403 - Services/                         (directory listing blocked)
403 - web.config                        (blocked)
200 - mini-profiler-resources/results   ⟵ unauthenticated
```

[21:31:15]

A Apex

It finds an information leak, and claims only what it can see.

Apex follows the one unauthenticated 200 and lands on an exposed debugging endpoint. The endpoint leaks infrastructure details. Apex describes what is actually there, and nothing more.

```
[21:31:15] A debugging profiler is exposed and leaking infrastructure details: an internal
hostname, an internal IP address, and an external IP address, all reachable without
authentication.
[21:31:56] The profiler does not contain any database queries, so I will not claim that. But the
hostname and internal IP exposure is a clear information disclosure on its own.
```

```
$ curl -sk 'https://example.acme.com/v20/mini-profiler-resources/results?id=<uuid>'
-------------------------------------------------------------------------------------------------
"MachineName":    "web-app-svc-07"
"User":           "192.168.[redacted]  -  [public-ip-redacted]"
"CustomTimings":  null
```

[21:34:06]

A Apex

It proves real impact, and stays read-only doing it.

Apex enumerates the methods on the application's internal AJAX service by reading the JavaScript that calls them, then calls each one without credentials. The interesting one is HtmlPreviewQuery, used by the app's inline preview panels. It is routed at the method level rather than the page level, so the login redirect that protects every .aspx page does not apply to it.

```
[21:34:06] Multiple methods on the internal web service are accessible without authentication.
One returns live records, including employee and vendor data with names, email addresses, and
phone numbers. This is a severe authentication gap exposing sensitive data.
```

```
$ curl -sk -X POST https://example.acme.com/v20/Services/AJAX.asmx/HtmlPreviewQuery \
  -H 'Content-Type: application/json' \
  -d '{"queryType":"Employee","queryKey":"1"}'
-------------------------------------------------------------------------------------------------
{"d":{"Header":"Employee ID: [redacted]",
      "Html":"...Contact Name:  [redacted]
              Email:         [redacted]@[redacted]
              Business:      [redacted-phone]..."}}
```

Apex also noticed a method on the same service that can change data (UpdateWorkOrderPartQuantity). It did not call that one. The bypass is proven across record types without a single write.

[21:34:57]

A Apex

It will not report what it cannot prove.

Apex spent real effort on SQL injection against the login form and both web services. The probes returned empty results, not errors. Rather than dress that up as a finding, Apex wrote it down as a dead end.

```
[21:34:57] The SQL injection attempts returned empty results rather than errors, suggesting the
application is handling them gracefully, likely parameterized queries or input validation.
```

```
$ curl -sk -X POST .../v20/services/Lookup.asmx/GetLookupResults \
  -d '{"lookupTable":"Asset","lookupField":"AssetCode",
       "lookupOption":"","term":"a OR 1=1"}'
-------------------------------------------------------------------------------------------------
{"d":[]}
```

[21:35:42]

A Apex

It dispatches a second agent to verify, then checks your history.

Apex does not trust its own first result. It dispatches a separate agent to retest the finding from scratch. The second agent gets no notes, no evidence, no shared memory; only the hypothesis. It runs its own commands and returns a verdict. Apex then checks the result against prior testing history and skips anything already on file. Severity is set against the existing record, not a fixed scale.

```
[21:35:42] Dispatching an agent to revalidate the finding. The agent has no access to my notes,
my evidence, or my prior tool calls; only the hypothesis to verify.
[21:36:10] validation pass completed: confirmed
[21:36:43] Checking against prior history. A previous critical finding on a different endpoint
(PortalRequest) was already fixed, so this is not a duplicate, though the two are related. A
separate profiler-disclosure issue is already open, so I will skip it rather than report it
twice. Given the earlier issue was rated critical and this exposes employee data through a
different path, I will rate this high.
```

```
$ [dispatch agent: revalidate]  "independently re-test the unauthenticated
                                 POST to /v20/Services/AJAX.asmx/HtmlPreviewQuery"

[query customer-history]        prior findings: "unauthenticated access
                                 AJAX.asmx PII"
-------------------------------------------------------------------------------------------------
agent (revalidate)   →  CONFIRMED
                         "an independent unauthenticated POST returned
                          the same Employee record. Auth bypass is real."

customer-history     →  finding 10001   (critical, fixed)   related, distinct path
                        finding 10002   (low, open)         duplicate, skip
```

[21:41:22]

A Apex

It dispatches the writeup, under discipline.

Once the finding is confirmed and history is clear, Apex dispatches two more agents. One writes the finding into the portal schema. The other drafts the assessment narrative against a static lint that rejects vague paragraphs, hand-waved error responses, and unsourced claims before they reach a reviewer.

```
[21:37:34] Dispatching an agent to write the finding for delivery. Schema: title, severity,
description, proof_of_concept, references. Severity 'high', not 'critical', given the
PortalRequest precedent.
[21:39:24] Dispatched the report writer to draft the assessment narrative. Lint flagged 4
consecutive prose paragraphs at line 1. The agent collapsed them into 2 paragraphs around a
request/response pair.
[21:40:26] Lint flagged enumerated negative results. The agent rewrote it as: SQL injection
testing on both the login form and the Lookup.asmx service produced no signal.
```

```
$ [dispatch agent: write-finding]   format the confirmed finding for delivery
                                    schema: title, severity, description,
                                            proof_of_concept, references

[dispatch agent: write-report]      draft the assessment narrative
                                    lint: no `...` inside code blocks
                                          no enumerated negative results
                                          no 4+ prose paragraphs without an anchor

-------------------------------------------------------------------------------------------------
agent (write-finding)   →  written, schema-valid
agent (write-report)    →  draft v4, 0 lint issues remaining
                           (3 paragraphs consolidated, 5 truncations marked,
                            1 negative-result line rewritten)
```

Sprocket Pentester

A Sprocket pentester reviews and signs off.

Apex takes the first pass; a member of the Sprocket testing team reviews and signs off on every finding before delivery. The same loop runs across your applications on a continuing basis.

HUMAN SIGN-OFF

Reviewed Apex's confirmed finding and its proof, independently re-tested the unauthenticated access, and signed off for delivery to customer. Severity held at high.