

**Gymnázium, Praha 6, Arabská 14**

**Programování**

## **Maturitní práce**



**2025**

**Oliver Hurt, 4.E**

# Čestné prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne 28.4. 2025

podpis: .....

# Anotace

Práce popisuje fungování obousměrné integrace (synchronizace) mezi open-source nástrojem pro incident response IRIS a SIEMem Sentinel od Microsoftu. V úvodu definuje pojmy jako Microsoft Sentinel, SOC, IRIS, SLA. Následuje detailní popis řešení pomocí Logic Apps, včetně automatizačních pravidel v Sentinelu a nastavení tzv. managed identity. Dále se v práci věnuji detailnějšímu popisu platformy IRIS a menší analýze jejího kódu. Také popisuji práci nad rámec zadání a to počítání SLA u jednotlivých bezpečnostních incidentů. Finálně prezentuji výsledky práce a popisuji jak řešení nasadit. Všechn kód je dostupný online na Github - <https://github.com/gyarab/2024-4e-hurt-hauerteUnifiedPortal>

This paper describes inner workings of bi-directional integration between open-source tool for incident response IRIS and SIEM Sentinel from Microsoft. In the beginning terms such as Microsoft Sentinel, SOC, IRIS, SLA are defined. Also, detailed description of implementation using Logic Apps, including automation rules in Sentinel, and set up of so called managed identities is provided. More detailed description of IRIS platform and brief analysis of its code is also included. Furthermore this paper includes work completed beyond the initial goal - calculating SLA of individual cybersecurity incidents. Finally, results of work done and deployment guide are presented. All code is available online on Github - <https://github.com/gyarab/2024-4e-hurt-hauerteUnifiedPortal>

Die Arbeit beschreibt die Funktionsweise der Zwei-Wege Integration zwischen dem Open-Source-Tool IRIS Incident Response und SIEM Sentinel von Microsoft. In der Einleitung werden Begriffe wie Microsoft Sentinel, SOC, IRIS, SLA definiert. Darauf folgt eine detaillierte Beschreibung der Lösung mithilfe von Logic Apps, einschließlich Automatisierungsregeln in Sentinel und Einstellung einer sogenannten managed Identity. Dann beschäftige ich mich mit einer detaillierteren Beschreibung der IRIS-Plattform und kurzer Analyse ihres Codes.

Ich beschreibe auch die Arbeit über die Lösung hinaus, nämlich die Berechnung von SLAs für einzelne Sicherheitsvorfälle. Abschließend stelle ich die Ergebnisse der Arbeit vor und beschreibe, wie die Lösung eingesetzt werden kann. Der gesamte Code ist online auf Github verfügbar - <https://github.com/gyarab/2024-4e-hurt-hauerteUnifiedPortal>

# Zadání

Zadání mého ročníkového projektu znělo následovně.

„Cílem mé ročníkové práce bude připojit se k vývoji open source nástroje DFIR-IRIS. DFIR-IRIS je ticketovací systém (podobný Jira) vyvíjený pro incident response. Moje práce se bude zabývat tvorbou stand-alone modulu pro oboustrannou integraci DFIR-IRIS  $\leftrightarrow$  Microsoft Sentinel. Práce bude mj. zkoumat jaké to je připojit se k FOSS komunitě vývojářů. Také popíši problémy (a výhody) GPLv3 licence a potřeby údržby kódu (patchování, upgrade na novější verze, zpětná kompatibilita..).”

1. Úvod	6
2. Microsoft Sentinel	7
3. IRIS	9
4. Implementace pomocí Logic Apps	13
5. Počítání SLA - nad rámec zadání	25
6. IRIS v Azure a deployment	32
7. GPLv3 licence	35
10. Závěr	36
11. Zdroje obrázků	37

# 1. Úvod

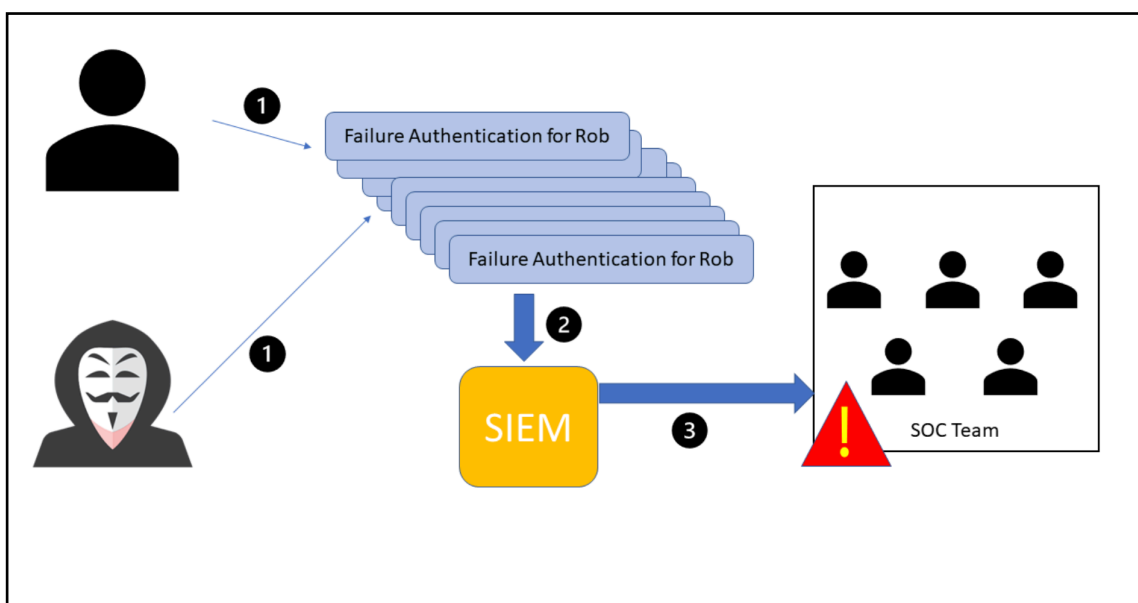
Cílem projektu bylo vytvořit **integraci** mezi free open source nástrojem (IRIS) a nástrojem z cloudového prostředí (Microsoft Sentinel) - prozkoumat, jak psát kód, který si bude rozumět s programem od velikánů jako Microsoft. Cílem kromě samotné funkčnosti bylo, aby bylo řešení pokud možno stabilní, tedy zvládalo produkční prostředí a zároveň aby bylo lehké přenositelné - aby ho mohli jednoduše replikovat další uživatelé.

V úvodu je potřeba ujasnit co to Microsoft Sentinel, IRIS je a proč by měla vzniknout integrace mezi nimi. Pokusím se to popsat v následujících kapitolách.

V poslední řadě bych chtěl upozornit na to, že práce sice obsahuje odhalené IP adresy, API klíče apod. které zde byly ponechány pro jednodušší pochopení, nicméně už jsou dávno zastaralé a neexistující.

## 2. Microsoft Sentinel

Microsoft Sentinel je cloud-native SIEM (Security Information and Event Management). SIEM je software pro zabezpečení, který organizacím poskytuje přehled o aktivitách v celé síti „z ptačí perspektivy“, aby mohly rychleji reagovat na hrozby – než dojde k narušení chodu firmy. Software, nástroje a služby SIEM detekují a blokuji bezpečnostní hrozby pomocí analýzy v reálném čase. Shromažďují data z celé řady zdrojů, identifikují aktivity, které se odchyľují od normálu, a provádí odpovídající akce<sup>1</sup>. V jednoduchosti SIEM je nástroj, který sbírá data ze všech endpointů (PC, severu...) ve firmě a po tom na základě dat hledá anomálie. Snaží se detekovat útoky. Pokud takovou anomálii nebo útok detekuje, vytvoří tzv. bezpečnostní incident. A právě pomocí Microsoft Sentinel můžou bezpečnostní analytici (zaměstnanci tzv. SOC - Security Operations Center) incidenty detekovat a adekvátně na ně reagovat.



Obr. č. 1 - SIEM use-case

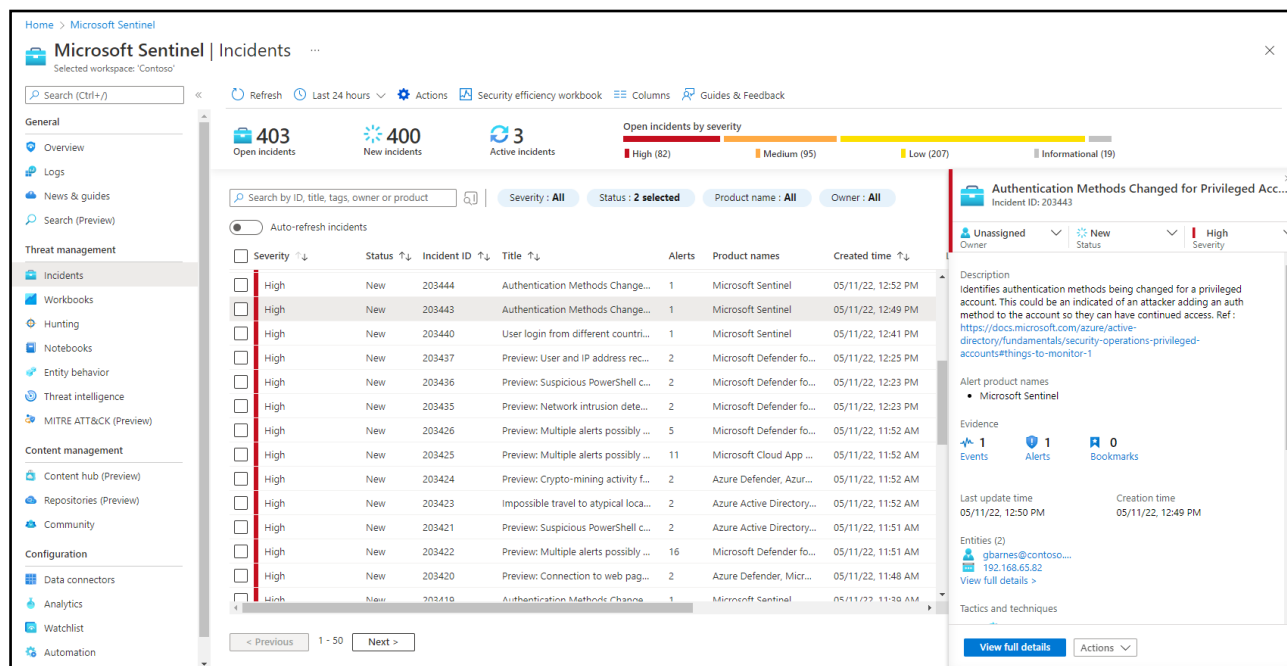
Uvažme příklad kdy zaměstnanec Rob provede (neúspěšně) 10 pokusů o přihlášení (Obr. č. 1). Mohlo by se jednat o legitimní pokusy, kdy jen Rob zapomněl heslo, nebo by mohlo jít o útočníka snažícího se dostat do účtu brute-force útokem. V jakémkoliv případě se ale zmíněných 10 pokusů o přihlášení posílá do SIEMu, který událost vyhodnotí jako anomálii a vytvoří incident<sup>2</sup> typu „possible brute force attack on account Rob“ na který budou následně zaměstnanci SOCu reagovat. Členové SOCu se pustí do *investigace* bezpečnostního incidentu. Pokud např. zjistí, že pokusy o přihlášení pochází z Robovi běžné IP adresy, z jeho firemního zařízení, mohou si být jistí že se o brute-force attack nejedná a incident zavřít jako legitimní aktivitu uživatele<sup>3</sup>. V opačném případě mohou členové SOCu např. blokovat IP adresu útočníka a doporučit multifaktorové ověřování (MFA).

<sup>1</sup>Autor neuveden. MICROSOFT.COM. Co je SIEM? [online]. [cit. 27.3.2025]. Dostupný na WWW: <https://www.microsoft.com/cs-cz/security/business/security-101/what-is-siem>

<sup>2</sup> To jak se „vytváří“ incident na základě sbíraných dat je mimo rozsah této práce, nicméně se jedná o tzv. *analytická pravidla*, která se spouští automaticky v pravidelných intervalech (např. 1 za den). Analytické pravidlo je (v případě Sentinelu) KQL query, která se spouští nad sbíranými daty a pokud vygeneruje výsledek - tedy zachytí anomálii - vytvoří incident. Více třeba zde: <https://learn.microsoft.com/en-us/azure/sentinel/create-analytics-rules?tabs=azure-portal>

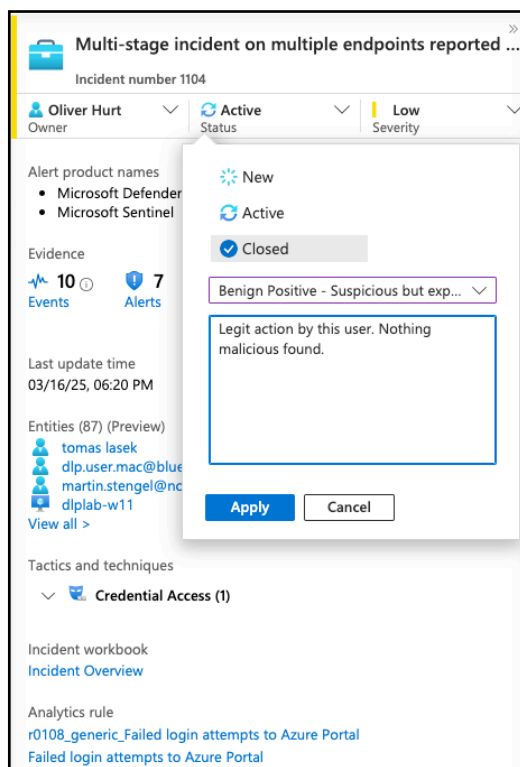
<sup>3</sup> Je vhodné si zde uvědomit, že reset hesla uživatele není v kompetenci SOCu nýbrž IT administrátorů dané firmy. Tzn. SOC se stará pouze o bezpečnost.

Na následujícím obrázku (Obr. č. 2) můžeme vidět jak vypadá Microsoft Sentinel v praxi.



Obr. č. 2 - Microsoft Sentinel

Každý řádek v hlavní části obrazovky odpovídá jednomu incidentu. Kliknutím na daný incident ho vybereme a zobrazí se jeho details v panelu na pravo. Bezpečnostní analytici - zaměstnanci SOCu si můžou incident přiřadit, změnit jeho stav (např. active), změnit jeho severitu (závažnost), případně přidávat komentáře a po konci investigace incident zavřít.

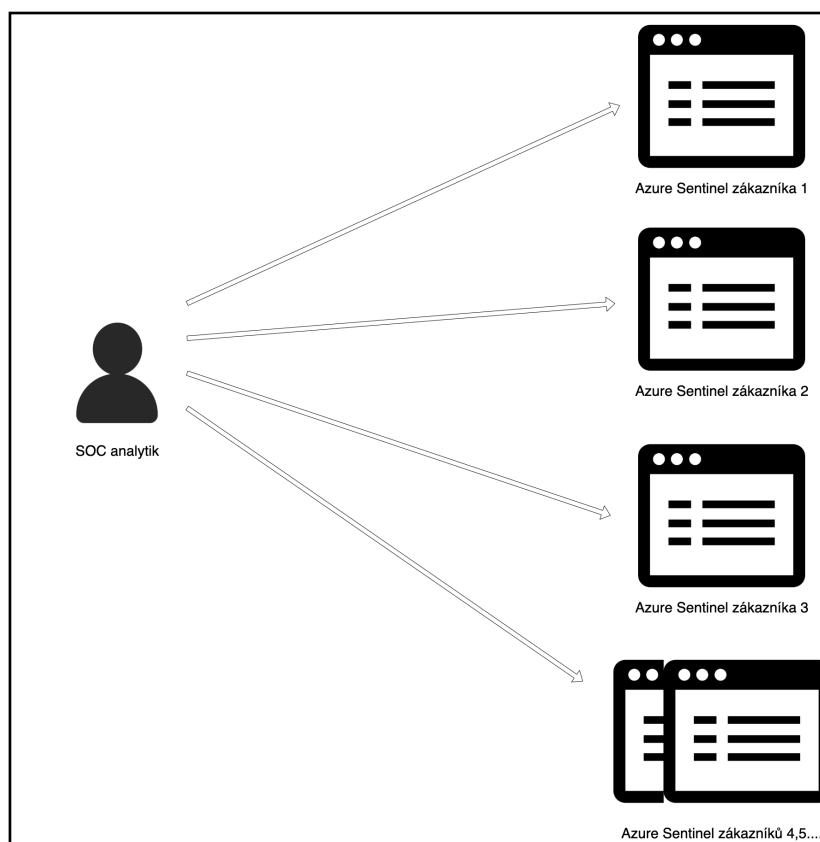


Obr. č. 3 - detail při zavírání incidentu po skončení investigace



### 3. IRIS

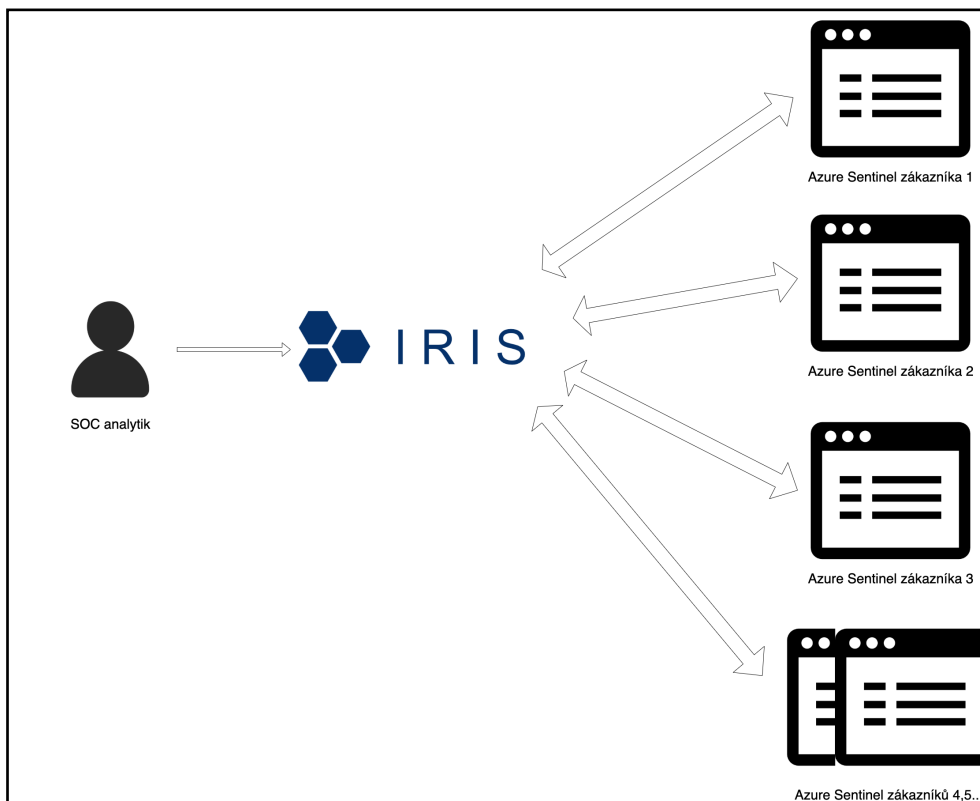
Tato kapitola obsahuje popis platformy IRIS. Předtím, než si ji ale datelněji popíšeme, nastíním stávající situaci - SOC bez platformy jako je IRIS



Obr. č. 4 - práce SOC analytika bez systému pro centrální správu incidentů

Při každodenní práci SOC analytik využívá Microsoft Sentinel pro řešení incidentů, problém ale nastává v případě, kdy SOC roste a stará se o bezpečnost více firem. Každý zákazník má svá specifika a vyžaduje vlastní instanci Sentinelu. Aby SOC mohl efektivně pracovat a odpovídat na bezpečnostní incidenty, musí být schopen je všechny sledovat. V momentě, kdy nemáme k dispozici žádný centrální systém pro správu incidentů, musí analytik v prohlížeči sledovat několik oken najednou, což je nejen nepraktické, ale při větším počtu zákazníků dokonce neproveditelné. Navíc ve firmě, kterou zaměstnanci SOCu brání a aktivně monitorují, pracují i jejich vlastní IT odborníci přes bezpečnost, kteří také sledují Sentinel a jsou schopni omezeně reagovat na bezpečnostní incidenty. Pokud takovou akci provedou (např. přidají komentář, či zavřou incident) je potřeba tuto změnu reflektovat do IRIS.

Právě z tohoto důvodu vznikají systémy pro centrální správu incidentů - IRIS je jedním z nich. Na následném obrázku je vidět opět práce z pohledu SOC analytika, tentokrát ale s platformou IRIS (Obr. č. 5).



Obr. č. 5 - IRIS - centrální systémem správy incidentů

Nyní analytik nemusí na přímo sledovat několik instancí Sentinelu, ale stačí sledovat jeden portál, kde přehledně vidí incidenty všech zákazníků a může na ně reagovat. Workflow incidentu bude tedy vypadat následovně:

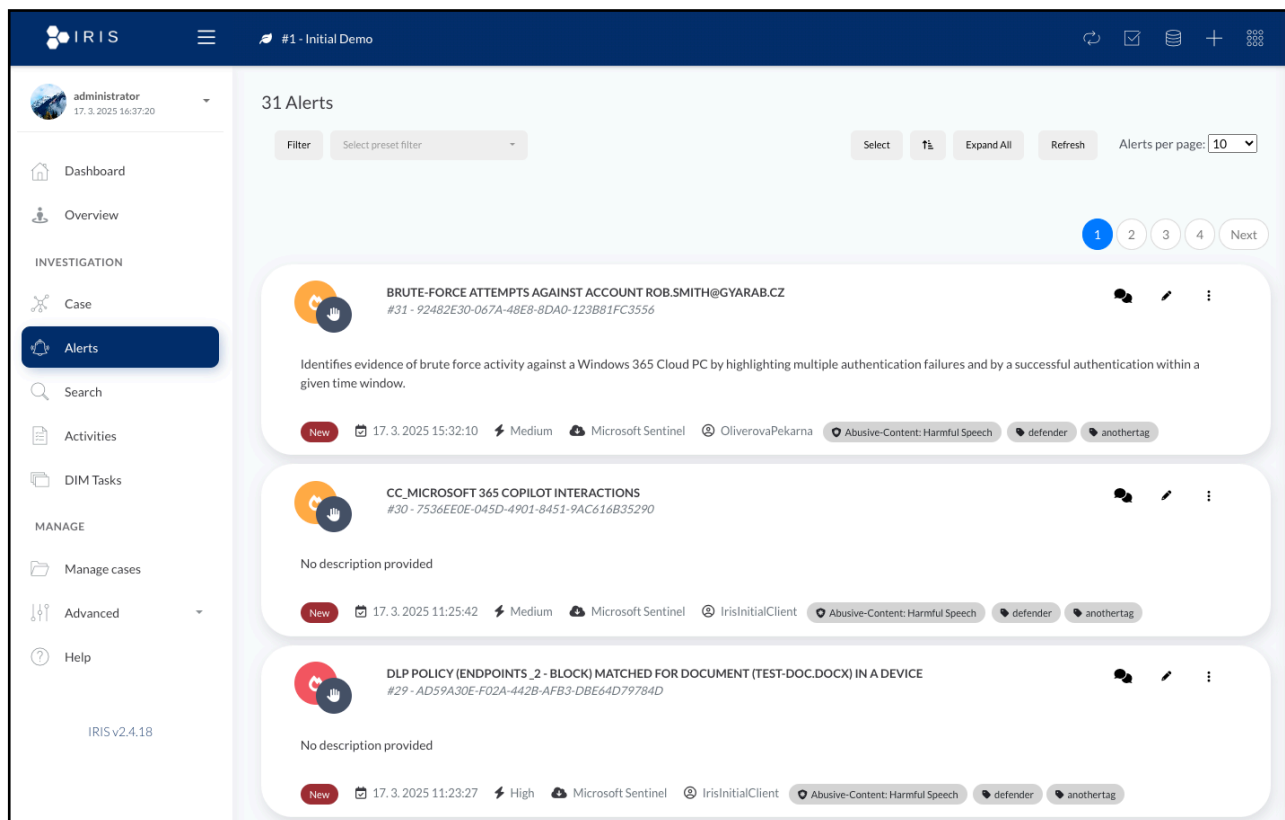
1. incident vznikne v Sentinelu zákazníka č. 1
2. Incident se pošle do platformy IRIS, kde se zobrazí
3. Analytik provede investigaci a v IRISu si přiřadí incident na sebe, přidá komentáře, zavře s výsledkem investigace incident.
4. Změny provedené v IRISu se synchronizují do Sentinelu zákazníka č. 1

Právě obousměrná **synchronizace mezi platformou IRIS a Microsoft Sentinel** byla **předmětem mé práce**. Je třeba podotknout, že analytik pořád může přiřazovat incident, měnit jeho stav, zavírat incident i přímo ze Sentinelu. Je tedy potřeba, aby integrace IRIS<—>Sentinel byla opravdu obousměrná. Každá změna v incidentu provedená v Sentinelu se propíše/projeví v IRIS, a opačně - každá změna v IRIS se projeví v Sentinelu. Právě zmíněná obousměrná integrace je na obrázku č. 5 představena obousměrnými šipkami.

Jak jsem již dříve uvedl, IRIS je centrální systém pro správu incidentů. IRIS umožňuje kromě správy incidentů (alertů), i generování reportů, správu indikátorů kompromitace, vytváření timeline (časové osy) incidentu, zobrazení grafu entit u incidentu a další funkcionalitu pomocí modulů třetích stran. IRIS vznikl roku 2019 pod hlavičkou Airbus Cybersecurity ve Francii a postupně v prosinci 2021 přešel v open-source pod licencí LGPL3<sup>4</sup>. Moje ročníková práce je fork původního projektu.

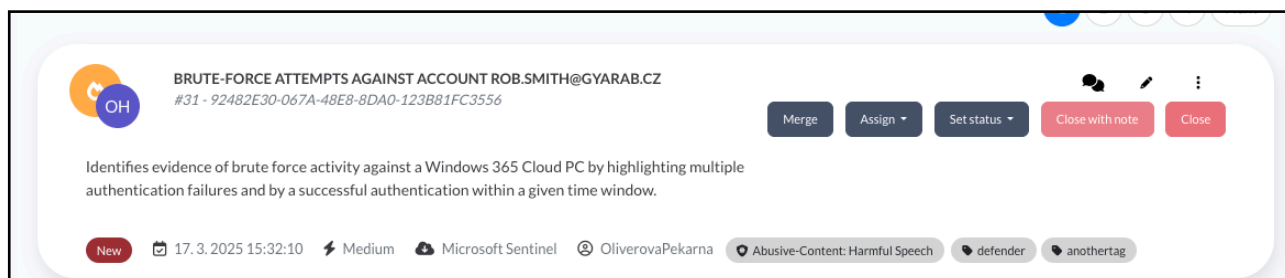
<sup>4</sup> KERNEL. DFIR IRIS. A mid-year review [online]. [cit. 23.2.2025]. Dostupný na WWW: [https://blog.dfir-iris.org/blog/iris\\_mid\\_year\\_summary/](https://blog.dfir-iris.org/blog/iris_mid_year_summary/)

Na obrázku č. 6 můžeme vidět incidenty z portálu IRIS.



Obr. č. 6 - Alerty/incidenty v IRIS

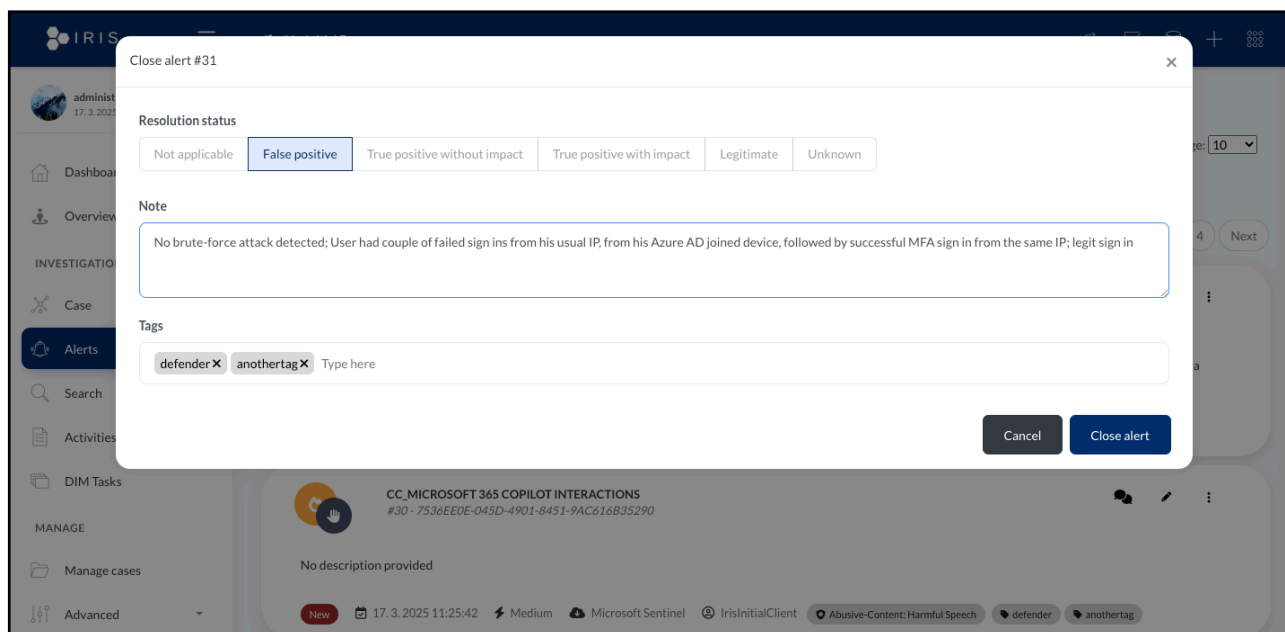
Přejetím myši na daný alert se zobrazí tlačítka pomocí kterých si můžeme incident přiřadit, změnit stav, zavřít... (Obr. č. 7)



Obr. č. 7 - možnosti u alertu

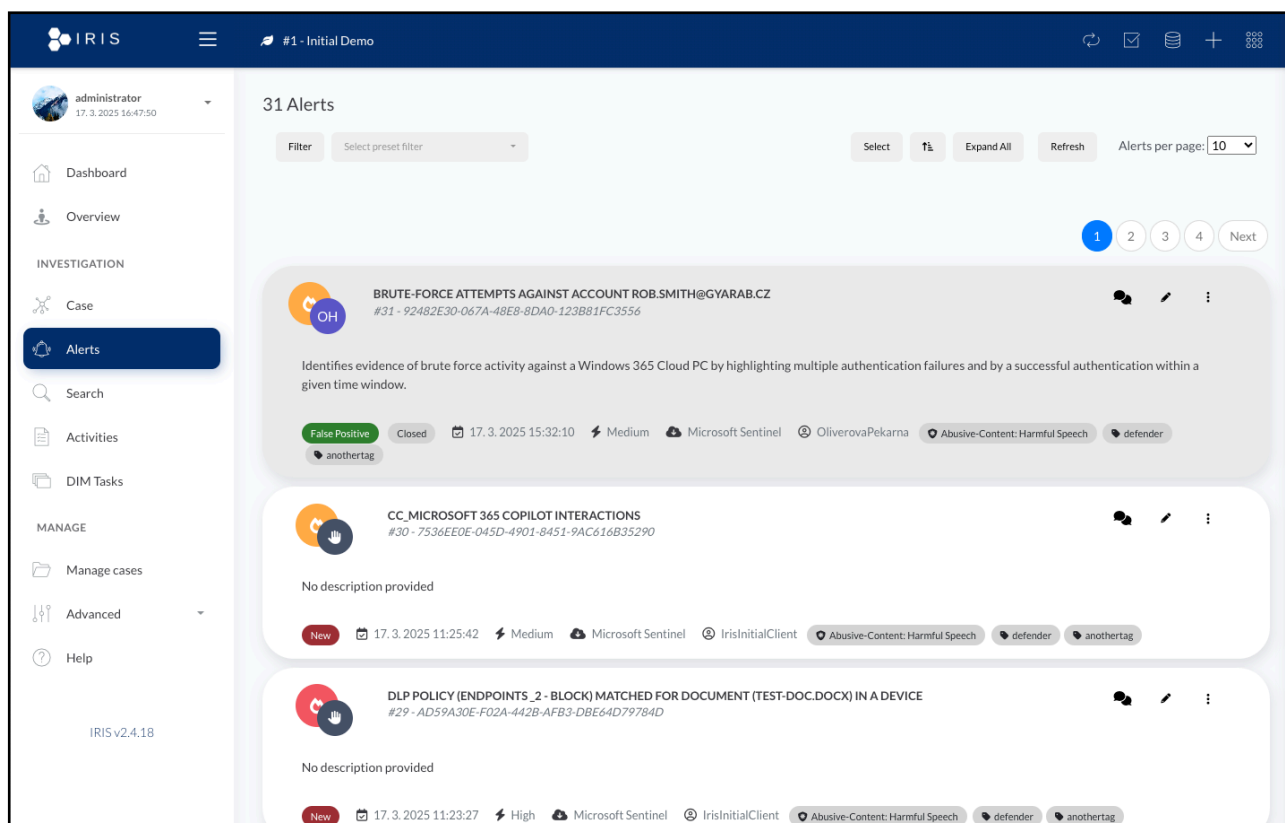
**Zde je třeba upozornit na to, že to, že zde incidenty vůbec jsou, je až výsledkem mé práce, nikoliv prerekvizitou. To jak se incidenty z IRIS do Sentinelu dostaly, je předmětem práce a je popsáno níže v dokumentu.**

Následně po skončení investigace můžeme incident zavřít pomocí tlačítka *close with note*. (Obr. č. 8)



Obr. č. 8 - detail při zavírání incidentu po skončení investigace

A následně výsledek v UI. (Obr. č. 9)



Obr. č. 9 - alert zavřený jako false positive

## 4. Implementace pomocí Logic Apps

Azure Logic Apps je cloud-native platforma, která umožňuje vytvářet a používat automatizační workflows. Workflow je něco jako tzv. *cloud function* nebo také *serverless function* - slouží většinou k jednomu účelu, běží čistě v cloudu (proto serverless) a jen na dotaz (on-demand). Platí se za výpočetní čas běhu Logic app. Každá Logic App respektive její workflow má nějaký *trigger* (spouštěč) - může to být jednoduchý HTTP request, email, událost v google kalendáři nebo jakýkoliv jiná událost. Právě proto, že Logic Apps podporují stovky různých *triggerů*, včetně těch pro události ze Sentinelu, byly vybrány jako nejlepší cesta pro implementaci. Jelikož jsou Logic Apps cloud-native platforma Microsoftu; přímo běžící v Azure (stejně jako Sentinel) dá se předpokládat i jistá *robustnost* řešení. Dále také bezpečnost a velmi efektivní škálovatelnost (scalability) a přenositelnost. V neposlední řadě, psát vlastního klienta pro dotazy na Microsoft Sentinel by bylo nejspíše nejen neefektivní, méně bezpečné a časově dražší.

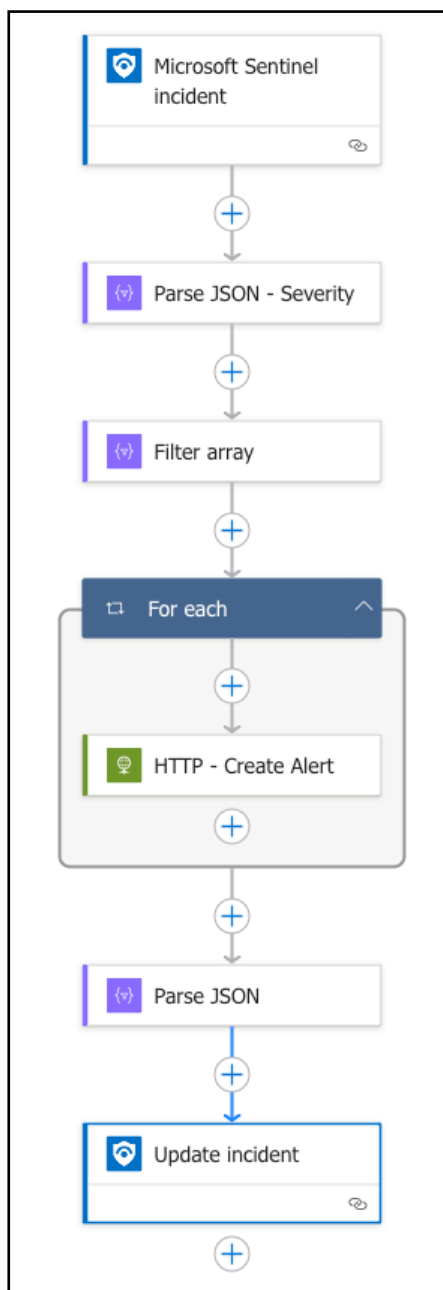
Jak již bylo avizováno např. obousměrnými šipkami na Obr. č. 5, cílem byla obousměrná synchronizace (nebo integrace chcete-li) mezi Microsoft Sentinel a platformou IRIS. Jednotlivé Logic Apps, které se o integraci starají jsou zobrazeny na Obr. č. 10.



Obr. č. 10 - Logic Apps starající se o integraci

Zelenými šipkami jsou znázorněny události (events) směrem ze Sentinelu do IRIS a modrými šipkami události směrem z IRIS do Sentinelu. Např. (Zelenou šipkou) pokud vznikne incident v Sentinelu, spustí se (díky triggeru) Logic App jménem **Iris-Sync-Incidents**, která pošle HTTP request na API IRISu, který tam vytvoří daný incident (dle terminologie IRISu Alert).

Následuje detailnější popis Logic App Iris-Sync-Incidents. (Obr. č. 11). Iris-Sync-Incidents se stará o to, aby když vznikne incident v Microsoft Sentinel, vznikl i v IRIS.



Obr. č. 11 - Logic App Iris-Sync-Incidents

První událost v Logic App je trigger „Microsoft Sentinel Incident“. Trigger je „připojený“ ke konkrétní instanci Sentinelu, ale o tom později. Po tom co se vytvoří incident v Sentinelu, dostane akce Microsoft Sentinel Incident všechny data o daném incidentu - viz obrázek č. 12

Microsoft Sentinel incident

Parameters Settings Code view About

Workspace ID

c887ac76-a09f-43c1-ad62-7dce977fe89a

object

```
{
  "id": "/subscriptions/6b7f39e0-3188-45e6-9e0f-d0c900e9c2b7/resourceGroups/rg-sentinel/providers/Microsoft.OperationalInsights/workspaces/law-sentinel/providers/Microsoft.SecurityInsights/Incidents/0c06cc28-33f4-4874-9621-367aa6a48185",
  "name": "0c06cc28-33f4-4874-9621-367aa6a48185",
  "etag": "\"b4017432-0000-0d00-0000-67d95e110000\"",
  "type": "Microsoft.SecurityInsights/Incidents",
  "properties": {
```

PROPERTIES

Obr. č. 12 - data konkrétního incidentu

Potom proběhne parsing HTTP dat a vybere se pole severity (závažnost) - viz Obr. č. 13. To děláme, protože potřebujeme hned při vzniku incidentu vědět, jakou severitu potřebujeme nastavit v IRIS. Pokud v Sentinelu vznikne LOW (nízká závažnost) musíme vytvořit LOW, pokud ale vznikne HIGH sev, musíme vytvořit HIGH sev.

Parse JSON - Severity

Filter array

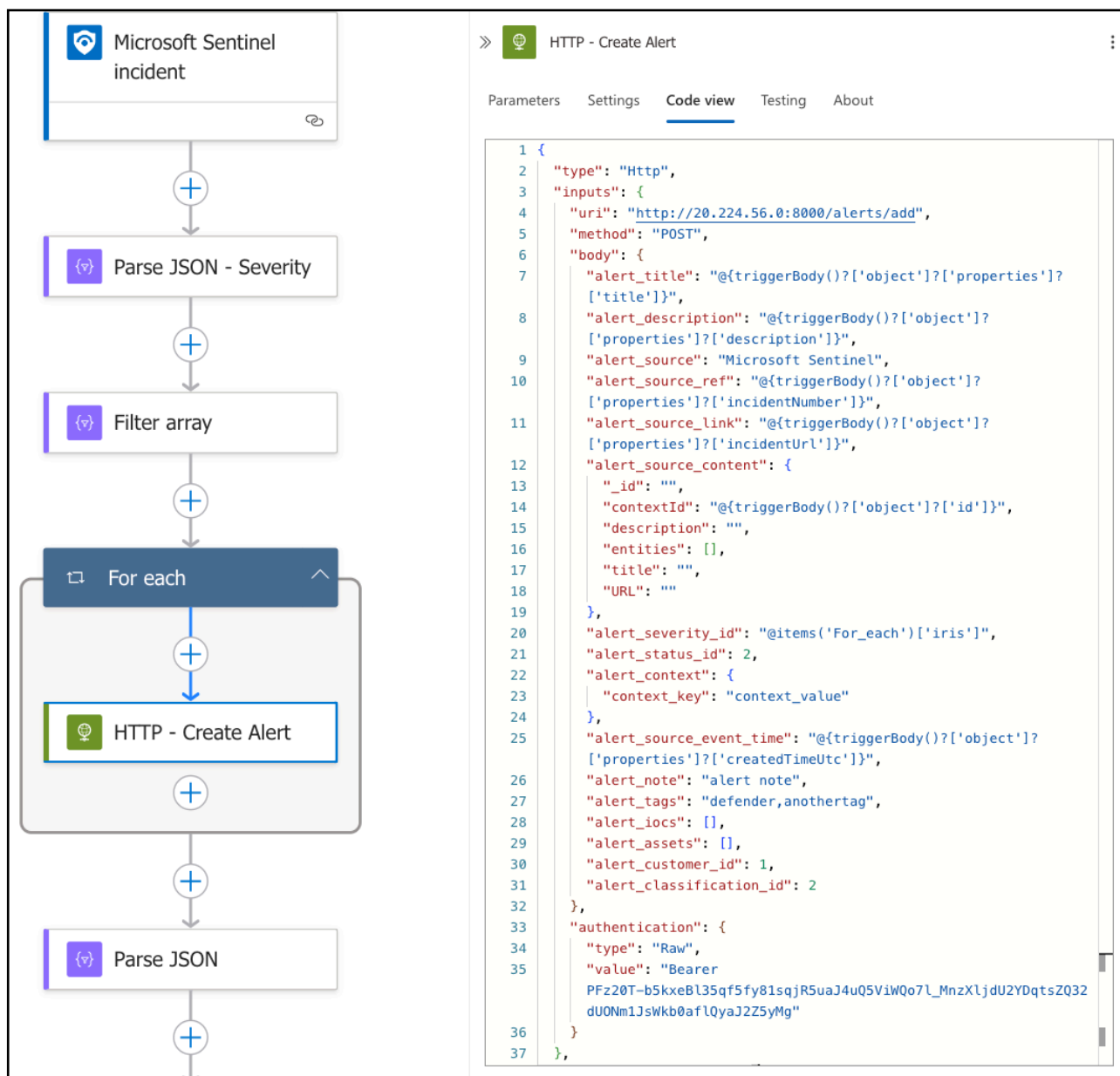
For each

Parameters Settings Code view Testing About

```
1 {
2   "type": "Query",
3   "inputs": {
4     "from": "@body('Parse_JSON_-_Severity')",
5     "where": "@equals(item()['sentinel'],triggerBody()['object']?['properties']?['severity'])"
6   },
7   "runAfter": {
8     "Parse_JSON_-_Severity": [
9       "Succeeded"
10    ]
11  }
12 }
```

Obr. č. 13 - filter array - severity

Následuje HTTP request na IRIS který vytvoří incident/alert. Viz Obr. č. 14.



Obr. č. 14 - HTTP request vytvářející alert v IRIS

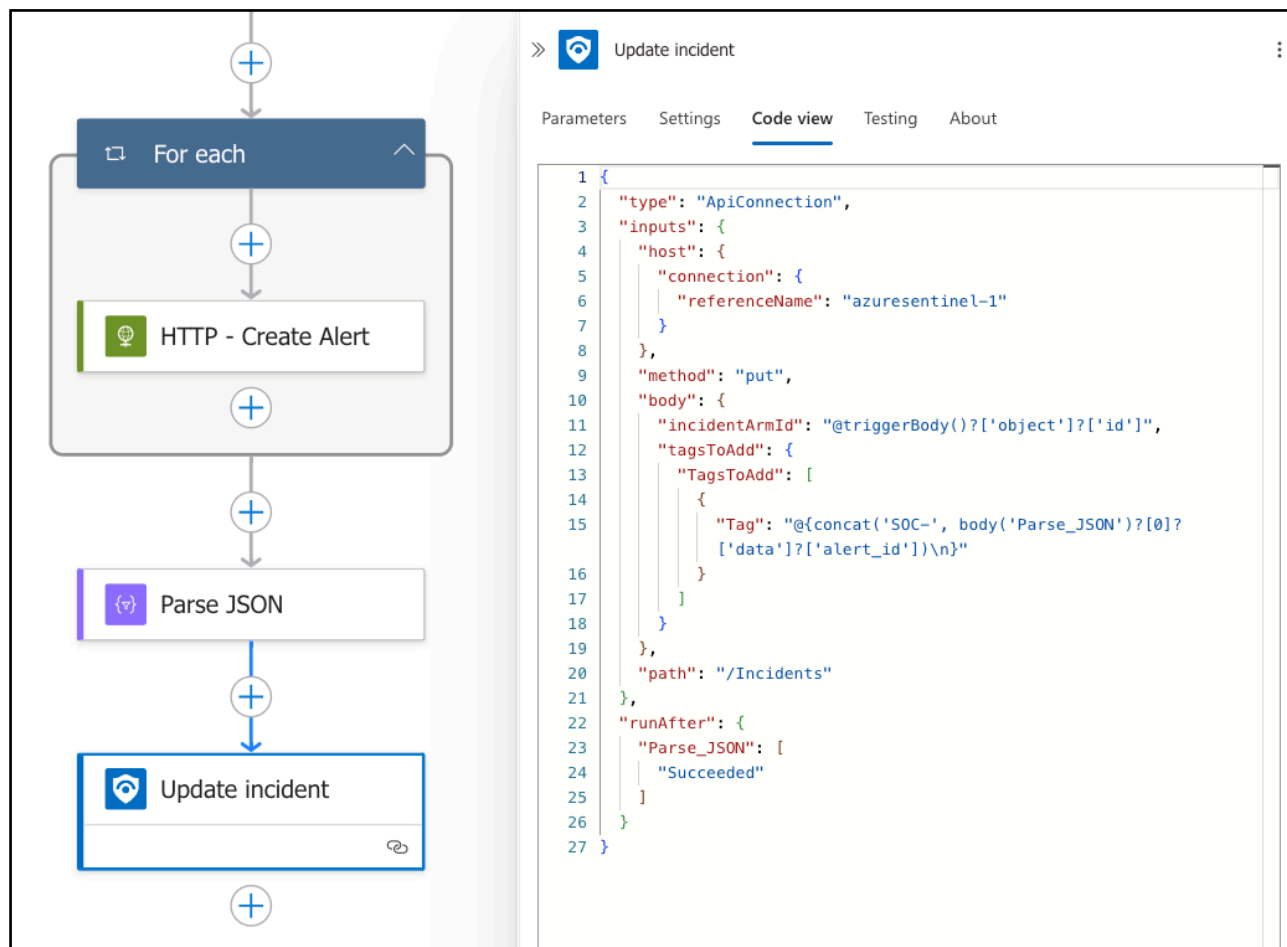
Dynamicky se ze Sentinelu do IRIS posílá: název incidentu (incident title), popis incidentu (description), ID incidentu v Sentinelu, URL incidentu, ARM ID incidentu (unikátní ID), severita (proto ten předcházející parsing) a čas vzniku incidentu.

For cyklus je zde, protože u severity se musí projít seznam všech severit IRIS a přiřadit ta odpovídající ze Sentinelu.

Po tom co se request pro vytvoření alertu odešle, parsuje se odpověď od IRIS serveru - ta je také HTTP request, který obsahuje všechny pole právě vzniklého alertu a dále status „success“, pokud se operace podaří, případně „failure“ a error hlášku. Jedním z atributů, který nám IRIS posílá zpět je *alert\_id* daného alertu, přiřazené IRISem. To děláme z toho důvodu, že pokud máme vícero instancí Sentinelu, může se stát, že u dvou zákazníků vznikne incident se stejným ID v sentinelu - vznikl by konflikt. Proto, necháváme IRIS vygenerovat centrálně unikátní *alert\_id*, které potom pomocí tagu přiřadíme incidentu v Sentinelu, a můžeme se na něj přistě odkazovat.

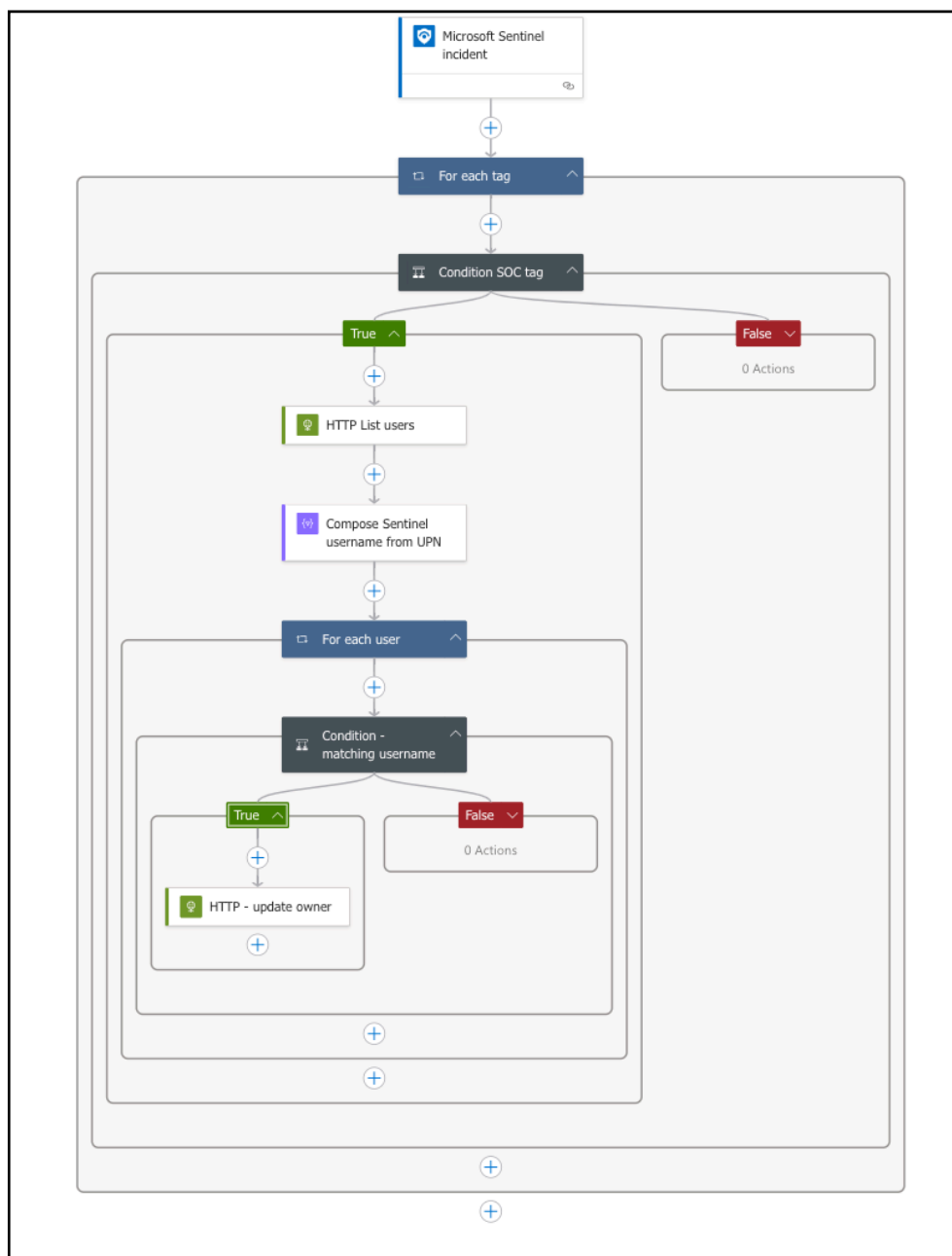


Nakonec jen tedy, pro úplnost akce Update incident na obrázku č. 15.



Obr. č. 15 - Update incident - přidání tagu s IRIS alert\_id

V podobném duchu funguje i Logic App **Iris-Sync-Owner** viz. obr. č. 16. Ta obsluhuje přiřazení vlastníka incidentu. Pokud si v Microsoft Sentinel přiřadím incident na můj účet oliver.hurt, musí se tato změna reflektovat i v IRIS.



Obr. č. 16 - LA IrisSyncIncidents

Trigger je opět změna v Sentinelu. Na začátku zkontrolujeme, že má incident tag ve tvaru „SOC-X” (kde X je ID alertu). Jedná se o ten tag, který jsme přiřadili předtím pomocí Logic App Iris-Sync-Incidents. Pokud ano, pokračujeme HTTP requestem na IRIS se seznamem všech uživatelů, který následně porovnáváme s UPN (User Principal Name, tedy celé jméno účtu, např. [oliver.hurt@student.gyarab.cz](mailto:oliver.hurt@student.gyarab.cz)) vlastníka incidentu v Sentinelu. Seznam, který přijde z IRIS může vypadat následovně - (viz další strana)

```
[
  {
    "user_id": 2,
    "user_uuid": "2dd7fa9f-c0f2-4539-a059-4d6781cdcf05",
    "user_name": "Oliver Hurt",
    "user_login": "oliver.hurt",
    "user_email": "oliver.hurt@student.gyarab.com",
    "user_active": true,
    "user_is_service_account": false
  },
  {
    "user_id": 1,
    "user_uuid": "7994812d-1ae1-46f4-b179-ad4fd4bc5bea",
    "user_name": "administrator",
    "user_login": "administrator",
    "user_email": "administrator@localhost",
    "user_active": true,
    "user_is_service_account": false
  }
]
```

A z triggeru incidentu víme, že vlastníkem je uživatel s UPN oliver.hurt@student.gyarab.com. Projdeme tedy for cyklem seznam a najdeme dle shodného emailu odpovídající IRIS user\_id. Finálně jen pošleme HTTP request, který přiřadí alertu v IRIS správného vlastníka (správné user\_id).

Nyní se pojďme podívat na komunikaci směrem z IRIS do Sentinelu - ta spoléhá částečně na kód v IRIS a částečně na Logic Apps. Rozhodl jsem se využít modulu IrisWebhooks, pro nastavení webhooku, který bude posílat data do Sentinelu. Bohužel je projekt IRIS stále ve vývoji, a tak je dokumentace značně zastaralá. Bylo tudíž potřeba ručně procházet codebase IRIS a odvodit, jak správně modul použít<sup>5</sup>. Po prohledání funkcí jako *call\_modules\_hook* a dalších, jsem se ve finálním produktu dostal k následující konfiguraci (viz obr. č. 17).



Obr. č. 17 - konfigurace webhooku IRIS → Sentinel

Jak můžeme vidět, konfigurace obsahuje zdrojovou URL (momentálně IP). Webhook se spustí na základě události *alert\_update* nebo *alert\_commented*, které definují funkce *alerts\_update\_route* a *alert\_comments\_get*<sup>6</sup>. Dále konfigurace obsahuje cílovou URL a v těle HTTP requestu bude odesílat dotyčný (updated) alert - respektive jeho data v JSON formátu. Pokud se tedy jakékoliv změní stav alertu v IRIS (updatne se - změni se stav, vlastníci, přidá se komentář..) na pozadí proběhne HTTP request na danou URL.

URL vede na Logic App, pojmenovanou logicky, **Iris-HTTP-Proxy**. Proxy (česky zástupce) je potřeba, jelikož IRIS umí poslat request jen na jednu cílovou URL. Pokud ale máme několik zákazníků, potřebujeme aby se aktualizovala data pouze zákazníka, jemuž alert/incident „patří“. Např. Máme zákazníky A, B, C. Pokud zrovna pracujeme na alertu zákazníka B, potřebujeme aktualizovat data v instanci Sentinelu zákazníka B a žádném jiném. Z tohoto důvodu byla vytvořena Logic App proxy.

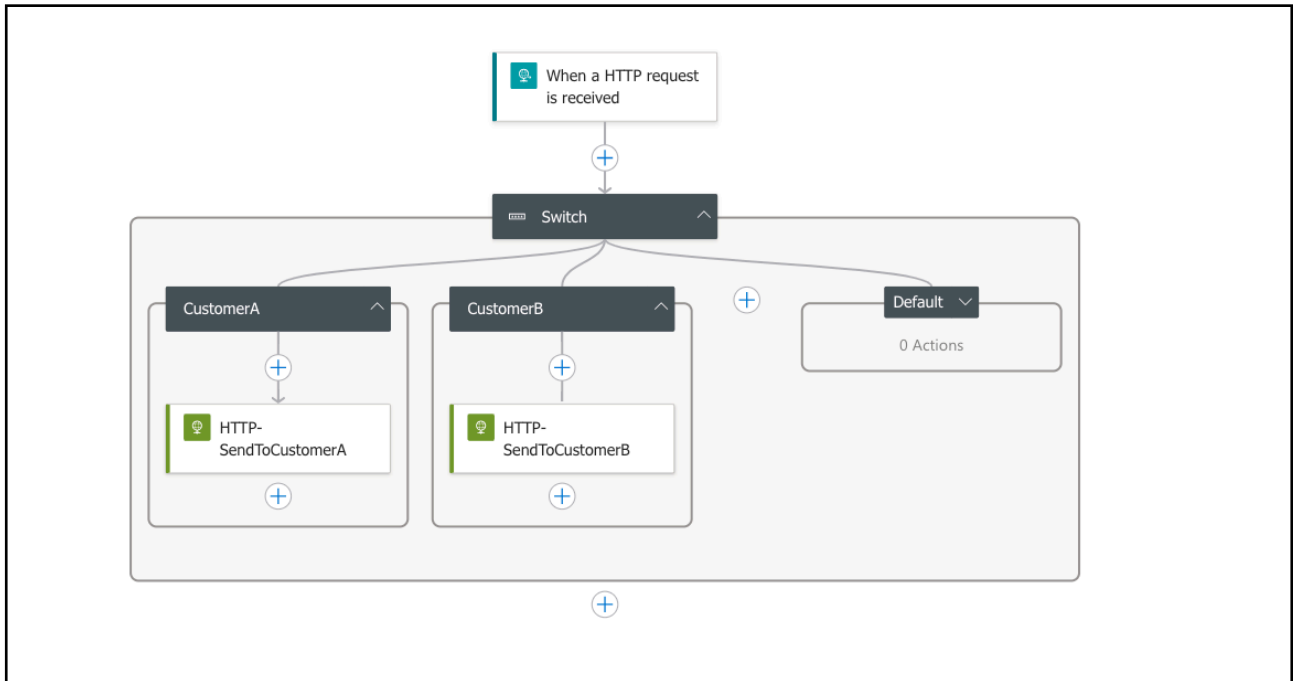
<sup>5</sup> K datu 9.1.2025 je stále oficiální ukázka konfigurace z dokumentace (<https://docs.dfir-iris.org/latest/operations/modules/natives/IrisWebHooks/>) nefunkční.

<sup>6</sup> Funkce jdou jednoduše najít pomocí funkcí Search Everywhere v PyCharm. (Nebo na Githubu)

Ačkoliv by bylo možné ručně upravovat kód IRIS tak, aby dokázal poslat HTTP request na více než jednu URL, byl by to proces delší a složitější, navíc by potencionálně mohl způsobovat merge konflikty při updatování na novější verzi z origin repozitáře. Navíc při větším počtu zákazníků je řešení proxy v LA mnohem přehlednější, než dlouhý JSON konfigurační soubor.

Logic App IRIS-HTTP-Proxy tedy dostane na vstup alert a na základě ID zákazníka (customer\_id) přepoše data správné Logic App daného zákazníka. Vždy se bude jednat o Logic App se jménem HTTP-Listener-<JménoZákazníka>.

Design IRIS-HTTP-Proxy je velmi jednoduchý a můžete ho vidět na obr. č. 18.

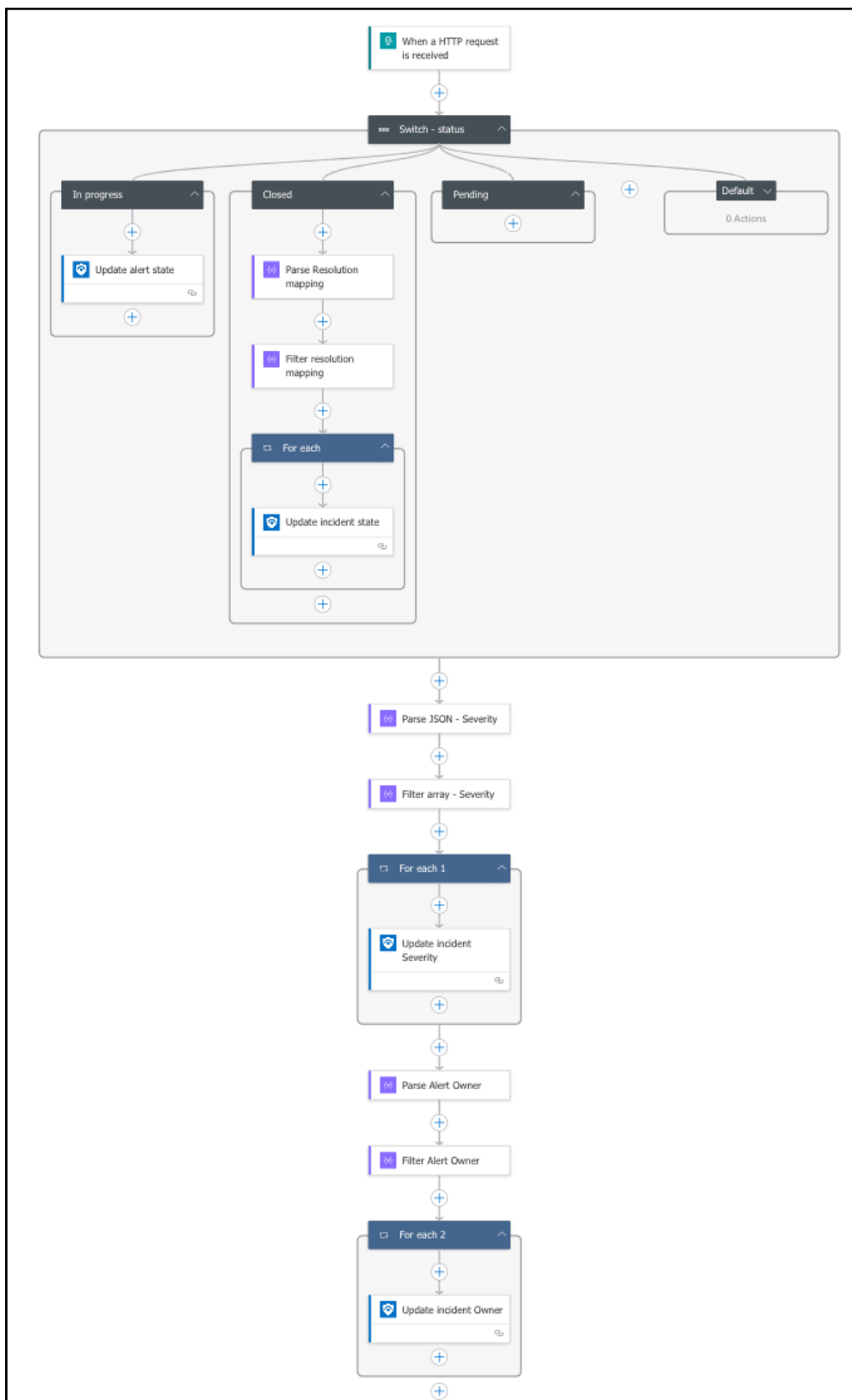


Obr. č. 18 - LA Iris-HTTP-Proxy

Akce HTTP-SendToCustomerA (i pro CustomerB) tedy přepoše původně získaná data z IRIS té logic App, která odpovídá danému zákazníkovi.

## IRIS-HTTP-Listener

Iris-HTTP-Listener je zodpovědný za všechny akce směrem IRIS —> Sentinel. (Popsáno např. na obrázku č. 10, str. 13). Jeho vstupem je alert z IRIS v JSON formátu, který mu přeposlala LA IRIS-HTTP-Proxy.



Obr. č. 18 - Iris-HTTP-Listener

Vypadá následovně - viz obr. č. 18. Na začátku se kontroluje změna stavu incidentu, která je u close trochu komplikovanější. Je to díky tomu, že IRIS a Sentinel nemají stejně klasifikované důvody pro zavření incidentu/alertu. Bylo tudíž potřeba namapovat odpovídající klasifikace. Aktuálně podle následující tabulky:

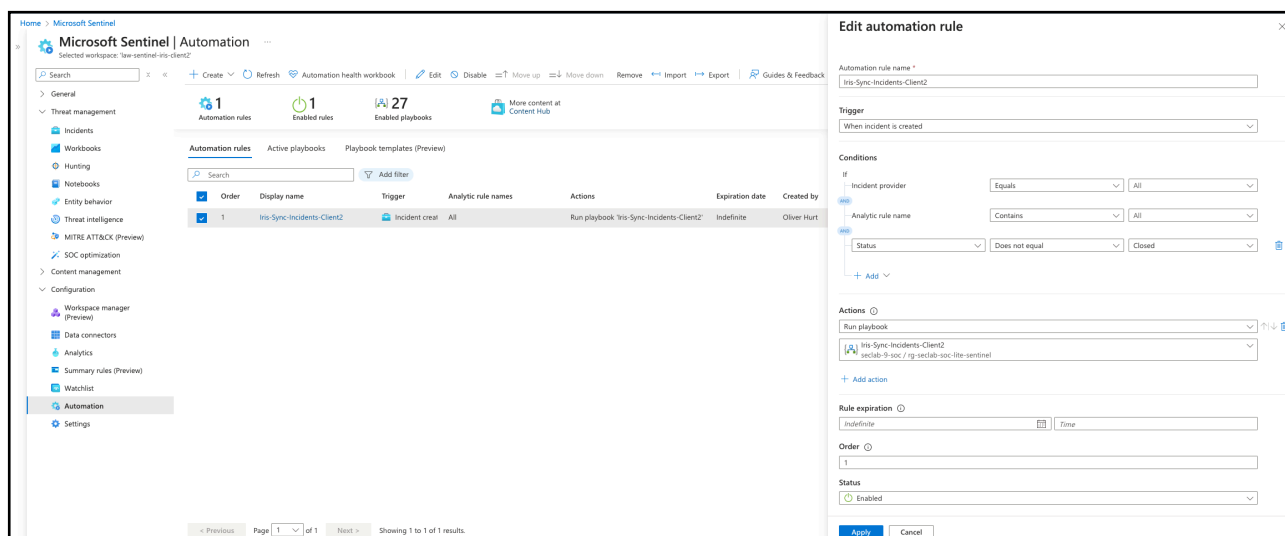
	Sentinel	Iris
1	Active	In Progress
2	Closed - benign positive	True Positive without impact
3	Closed - true positive	True Positive with impact
4	Undetermined	Closed
5	False Positive - Incorrect alert logic	False positive
6	False Positive - Inaccurate data	False positive

To také vysvětluje i proč je větev Pending prázdná - v Sentinelu žádný stav pending neexistuje. Dále se v Logic App upravuje severita a vlastník incidentu/alertu.

Není potřeba zde detailně popisovat implementaci každé Logic App - jejich kód ostatně najdete na Github repozitáři.

Nyní už víme jak fungují Logic Apps zařizující obousměrná integrace. Rychle se vrátím k propojení Logic App k dané instanci Sentinelu jak bylo avizováno na str. 15.

Jak jsem již uváděl, daná Logic App je vždy „vázaná“ k dané instanci Sentinelu. To, čím je „vázaná“, jsou takzvané Automation rules v Sentinelu. Pokud chceme, aby se daná Logic App spouštěla z našeho Sentinelu, musíme to nastavit. Konkrétně v Microsoft Sentinel > Configuration > Automation musíme vytvořit nové automatizační pravidlo (automation rule) - viz. Obrázek č. 19.



Obr. č. 19 - automatizační pravidlo Sync-Incidents v Sentinelu zákazníka č. 2

Automatizační pravidlo je zabudovaná funkcionality Sentinelu. Jak už název napovídá, pomáhá spouštět automatizace na základě událostí v Sentinelu. V tohoto konkrétním případě se pravidlo spustí pokaždé co vznikne nový incident v Sentinelu, který není ve stavu uzavřeno (closed). Pokud to platí, spustí Logic App nazvanou Iris-Sync-Incidents-Client-v2, která je identická k Iris-Sync-Incidents, kterou jsme si představovali dříve, akorát s tím rozdílem že Iris-Sync-Incidents-Client-

v2 se váže k Sentinelu zákazníka 2 (spouští ho automatizační pravidlo zákazníka č. 2, a ne automatizační pravidlo v Sentinelu zákazníka č. 1).

Aby Logic App ale běžela (mohla upravovat incidenty, číst je apod.), potřebuje dostat potřebná oprávnění. Aby je však mohla dostat, případně, aby je ostatní služby v Azure (např. Sentinel) mohli ověřovat, musí mít vlastní identitu. Identity se spravují pomocí Microsoft Entra ID a umožní nám mimo jiné používat RBAC (Role-based Access Control). Hlavně díky jednoduchosti při vývoji jsem se rozhodl použít takzvanou managed identitu. Managed identita je spravovaná; nemusíme se starat o ukládání nebo rotaci přihlašovacích údajů v kódu, protože Azure za nás spravuje veškeré autentizační procesy a bezpečnostní tokeny. Managed identita - v našem případě system-assigned managed identity<sup>7</sup> se nastavuje u dané Logic App. V nastavení Logic App (Settings) přejděte do Identity a zapněte System assigned managed identity a potom stačí přejít do Access Control (IAM) a přidat roli Microsoft Sentinel Responder. Následně nastavíme připojení do sentinelu pomocí právě vytvořené, oprávněné managed identity - podobě jako na obrázku č. 20.

Obr. č. 20 - trigger connection do Sentinelu pomocí managed identity

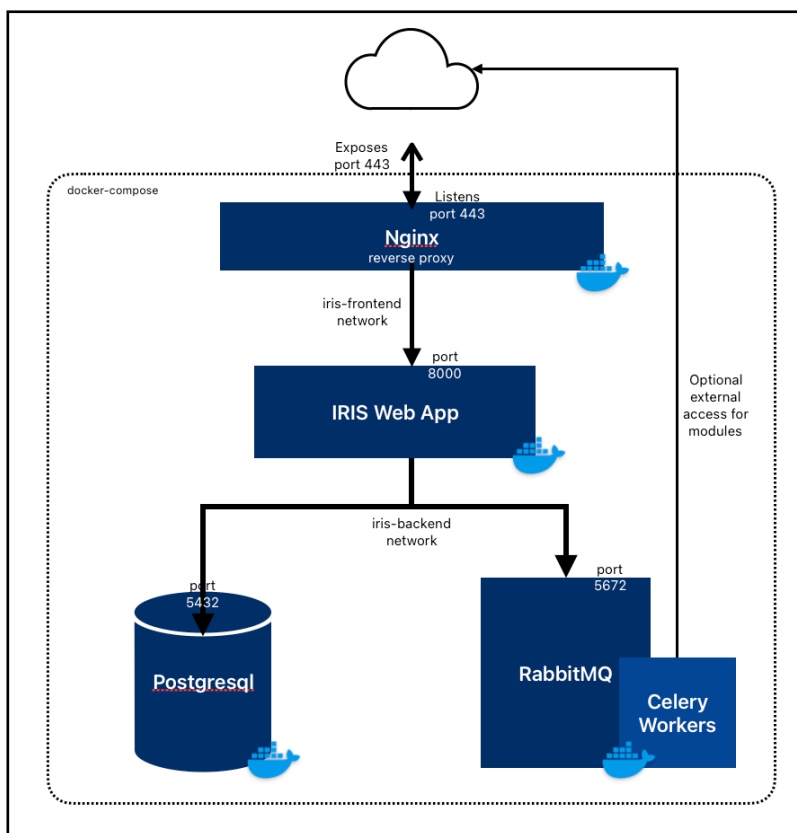
<sup>7</sup> Existuje ještě tzv. User-assigned managed identita, nicméně do detailů nejspíše nemá cenu v této práci zacházet. Více se můžete dočíst např. Zde: <https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/managed-identity-best-practice-recommendations>



## 5. Počítání SLA - nad rámec zadání

Service-level agreement (SLA) je termín, který označuje smlouvu sjednanou mezi poskytovatelem služby a jejím odběratelem. Většinou se SLA týká oblasti IT<sup>8</sup>. V našem případě definuje, jak rychle musí zaměstnanci SOCu reagovat na bezpečnostní incident. SLA se uvádí v hodinách - typicky je definováno 12h na reakci u incidentu s nízkou (low) severitou, 6h u střední (medium) severity a 1hod u incidentů s vysokou (high) severitou. Nejběžnější modely poskytování služby SOC jsou 24/7 nebo 12/5, záleží ale samozřejmě na dohodě se zákazníkem.

Jelikož počítání SLA je něco, co v IRIS chybí (jak dokazuje i např. Množství dotazů na tuto funkcionalitu v komunitě na Discordu IRIS), rozhodl jsem se ho dodělat. Možností řešení je pochopitelně několik - např. Posílat data přímo z Postgres databáze IRIS do Grafana, kde je výsledně vizualizovat, případně použít jiný vizualizační nástroj jako Power BI od Microsoft. Jelikož jsem ale chtěl, aby mohl analytik přehledně vidět stav SLA rovnou u daného incidentu/alertu, rozhodl jsem se pro implementaci přímo v codebase IRIS.



Obr. č. 21 - struktura IRIS

IRIS se skládá z 5 kontejnerů (viz. Obr. č. 21) - app, db, RabbitMQ, worker a nginx:

- app je jádrem aplikace a obsahuje zdrojový kód, startuje webserver
- Db je Postgres databáze
- RabbitMQ zpracovává a řadí úlohy do fronty (queue)
- worker obsluhuje RabbitMQ
- Nginx je proxy pro web requests

Jelikož začal IRIS používat pro front-end Svelte, musel jsem psát v něm, aby změny byly kompatibilní. Na back-end používá Python framework Flask.

<sup>8</sup> Autor neuveden. Service-level agreement [online]. [cit. 27.3.2025]. Dostupný na WWW: [https://cs.wikipedia.org/wiki/Service-level\\_agreement](https://cs.wikipedia.org/wiki/Service-level_agreement)

Na stránce /alerts se alerty *renderují* pomocí funkcí *refresh\_alerts* a *render\_alert*. Do obou funkcí jsem přidal kód v JS který vytvoří event kdykoliv jsou tyto funkce zavolány. Viz. Obrázek č. 22.

```
async function refreshAlert(alertId, alertData, expanded=false) {  
  
    document.dispatchEvent(new CustomEvent('alertRendered', {  
        detail: {  
            IRISTime: alertData.alert_creation_time,  
            alertStatusID: alertData.alert_status_id,  
            alertSevID: alertData.severity.severity_id,  
            alertCustomerID: alertData.alert_customer_id,  
            SLA: SLAdata,  
            alertID: alertData.alert_id  
        }  
    }));  
}
```

Obr. č. 22 - event alertRendered

Následně v souboru *SLA.js* poslucháme daný event (*alertRendered*) a pro každý div element s atributem class „*SLAcontainer*” v DOM injectneme Svelte komponent *SLAcontainer*. Komponentu *SLAcontainer* zároveň předáme vstupní data - čas vzniku alertu v IRIS, stav alertu, závažnost alertu, zákazníka alertu, data o SLA (o nich ještě později) a alert\_id z IRIS. Po injectování přidáme u div atribut class „*mounted*” abychom zabránili opakovanému mountování při obnovení (refresh) stránky. Viz Obr. č. 23.

```
import App from './SLAcontainer.svelte';  
  
export let app; // Declare at the top level Show usages ± Rumburaq2  
  
document.addEventListener( type: 'alertRendered', listener: (event) => { ± Rumburaq2*  
    const containers = document.querySelectorAll( selectors: '.SLAcontainer');  
    if (containers) {  
        containers.forEach( callbackfn: (container) => {  
            // Check if the container already has the "mounted" class  
            if (container.classList.contains("mounted")) {  
                console.log('SLA component already mounted on this container');  
                return;  
            }  
            app = new App({  
                target: container,  
                props: {  
                    IRISTime: event.detail.IRISTime,  
                    alertStatusID: event.detail.alertStatusID,  
                    alertSevID: event.detail.alertSevID,  
                    alertCustomerID: event.detail.alertCustomerID,  
                    SLA: event.detail.SLA,  
                    alertID: event.detail.alertID  
                }  
            });  
            console.log('Svelte component mounted:', app);  
            // Mark the container by adding a "mounted" class so we don't mount again.  
            container.classList.add("mounted");  
        });  
    }  
});
```

Obr. č. 23 - injekce Svelte komponentu do DOM

Komponent *SLAcontainer* jen předá data dalšímu komponentu *SLAbar*, a tak nemá cenu ho zmiňovat - ostatně kód je na Github. *SLAbar* se stará o většinu logiky - vypočítání SLA; respektive jeho začátku a konce. Komponent na začátku paruje SLA data, která dostal na vstup. Získá tak pole *workingDays* které obsahuje seznam všech dní kdy má SLA běžet a dva integery *startHour* a *endHour* které definují začátek a konec pracovní doby. Funkce *getClosestWorkingTime* má parametr *InputTime* a vrací true pokud je *InputTime* v pracovní době, jinak vrací nejbližší pracovní den a hodinu ve formátu DD/MM/YYYY HH:MM.

```
// Check if it's a working day and within working hours - if yes return true else return next working time
function getClosestWorkingTime(inputDateTime) { Show usages  ± Rumburaq2
  const inputDate = parseDateTime(inputDateTime);
  const currentDayIndex = inputDate.getDay(); // 0 for Sunday, 1 for Monday, etc.
  const currentHour = inputDate.getHours();
  // Convert day index to a string name
  const dayName = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
  // Check if it's a working day and within working hours
  if (
    workingDays.includes(dayName[currentDayIndex]) &&
    currentHour >= workingHours.start &&
    currentHour < workingHours.end
  ) {
    return true;
  }
  // Calculate the closest working time
  const closestTime = new Date(inputDate);
  // If it's outside working hours but on a working day
  if (workingDays.includes(dayName[currentDayIndex])) {
    if (currentHour < workingHours.start) {
      closestTime.setHours(workingHours.start, 0, 0, 0);
      return closestTime;
    }
    if (currentHour >= workingHours.end) {
      let daysToAdd = 1;
      while (!workingDays.includes(dayName[(currentDayIndex + daysToAdd) % 7])) {
        daysToAdd++;
      }
      closestTime.setDate(closestTime.getDate() + daysToAdd);
    }
  }
  else {
    // If it's not a working day, find the next working day
    let daysToAdd = 1;
    while (!workingDays.includes(dayName[(currentDayIndex + daysToAdd) % 7])) {
      daysToAdd++;
    }
    closestTime.setDate(closestTime.getDate() + daysToAdd);
  }
  // Set the time to the start of working hours
  closestTime.setHours(workingHours.start, 0, 0, 0);
  return closestTime;
}
```

Obr. č. 24 - funkce getClosestWorkingTime

Poslední Svelte komponent *ProgressBar* vykresluje finální podobu Progress baru pro zobrazení stavu SLA. Funkce *start* spustí lambda funkci *setInterval* která se bude sama volat každou sekundu. Uvnitř funkce se vypočítá aktuální čas v minutách od začátku Linux epochy. Pokud SLA stále běží (má default hodnotu -1) vypočítá se již uběhlý čas (*elapsed*). V opačném případě (načetla se nezáporná hodnota uplynulého SLA z databáze) se jen uběhlý čas nastaví na ten v databázi a zastaví volání funkce *setInterval*. A pokud je uběhlé SLA větší než celkový čas na splnění označíme SLA za brached. Analytik může zastavit - tedy splnit SLA tím, že dá incident do stavu *in Progress*. Je tomu tak, protože v našem případě se staráme o metriku Mean Time To Acknowledge - tedy čas za který *započne* reakce na incident, nikoliv jeho úplné vyřešení. Viz. Obrázek č. 25.

```
export let startDateTime = "01/02/2025 14:30"; Show usages  ± Rumburaq2
export let endDateTime = "01/02/2025 14:30"; Show usages  ± Rumburaq2
let timecomputed = calculateDuration(); //calculate duration between two timestamps in seconds
$: elapsed = 0;
$: duration = timecomputed;
let interval: number
let oldElapsedTime = 0;
let currTimeEpoch = 0;
let startTimeEpoch = getSecondsSinceEpoch(startDateTime);
$: SLAbreached = false;

function start() { Show usages  ± Rumburaq2 *
  interval = setInterval(() => {
    if(currstate === MyState.RUNNING) {
      //calculate curr time
      currTimeEpoch = Math.floor((Date.now()) / (1000 * 60)) * 60;
      if(dbElapsedSla < 0){ //SLA not yet completed
        elapsed = currTimeEpoch - startTimeEpoch + oldElapsedTime;
        console.log("elapsed: "+elapsed);
      }
      else { //SLA is not -1 therefore was completed sometime before
        elapsed = dbElapsedSla;
        clearInterval(interval)
        currstate = MyState.PAUSED;
      }

      if (elapsed > duration) {
        elapsed = duration
        clearInterval(interval)
        SLAbreached = true;
      }
      if (alertStatusID === 4){
        complete();
      }
    }
  }, 1000)
}
```

Obr. č. 25 - ProgressBar.svelte - funkce start

Funkce *complete* také zastaví volání funkce *setInterval* a zároveň запиše do databáze současnou hodnotu proměnné *elapsed*. Zapisování do databáze obsluhuje funkce *writeSlaData*. (obr. č. 26)

```
async function writeSlaData() { Show usages  ± Rumburaq2
  try {
    const response = await fetch(`alerts/api/set_elapsed_sla_api/${alertID}/${elapsed}`);

    const result = await response.json();
    console.log('Update result:', result);
  }
  catch (error) {
    console.error('Error fetching data:', error);
    throw error;
  }
}
```

Obr. č. 26 - funkce *writeSlaData*

Aby funkce *writeSlaData* fungovala (respektive mohla fungovat) bylo potřeba vystavit nový API endpoint. Tudiž vznikl v Flask back-endu route *alerts/api/set\_elapsed\_sla\_api* - viz. Obr. č. 27

```
@alerts_blueprint.route('/alerts/api/set_elapsed_sla_api/<int:alert_id>/<int:new_elapsed_sla>', methods=['GET'])
@ac_api_requires(Permissions.alerts_write)
def set_elapsed_sla_api(alert_id: int, new_elapsed_sla: int):
    updated_alert = set_elapsed_sla(alert_id, new_elapsed_sla)
    return response_success(data=updated_alert)

@alerts_blueprint.route('/alerts/api/get_clients_sla_api', methods=['GET']) ± Rumburaq2
@ac_api_requires()
def get_clients_sla_api():
    rows = get_clients_sla()
    customers = [dict(row._mapping) for row in rows]
    output = {
        "customers_sla": customers
    }

    return response_success(data=output)
```

Obr. č. 27 - route *alerts/api/set\_elapsed\_sla\_api*

Na obrázku č. 27 je kromě route která zapisuje do databáze (*set\_elapsed\_sla\_api*), pro ilustraci i ta, která z ní čte (*get\_clients\_sla\_api*).

Funkce *set\_elapsed\_sla* je odděleně, abychom v projektu oddělovali router od logiky databáze. Nicméně vypadá následovně (obr. č. 28).

```
def set_elapsed_sla(alert_id: int, new_elapsed_sla: int): ± Rumburaq2
    alert = Alert.query.filter(Alert.alert_id == alert_id).first()
    if alert:
        # Update the column with the new value
        alert.alert_elapsed_sla = new_elapsed_sla
        # Commit the changes to the database
        db.session.commit()
        return alert
    else:
        # Optionally handle the case where no alert is found
        return None
```

Obr. č. 28 - zápis do databáze

Finálně zobrazíme HTML element progress bar s aktuálními hodnotami. (Obrázek č. 29).

```
<div class="grid-gap">
  <div class="progress-container col-md-12" style="...">
    <div class="col-md-12" style="...">
      <div class="row col-md-12" style="...">
        <p>{startDateTime} - {endDateTime}</p>
      </div>

      <div class="row col-md-12" style="...">
        <span>Elapsed time:</span>
      </div>

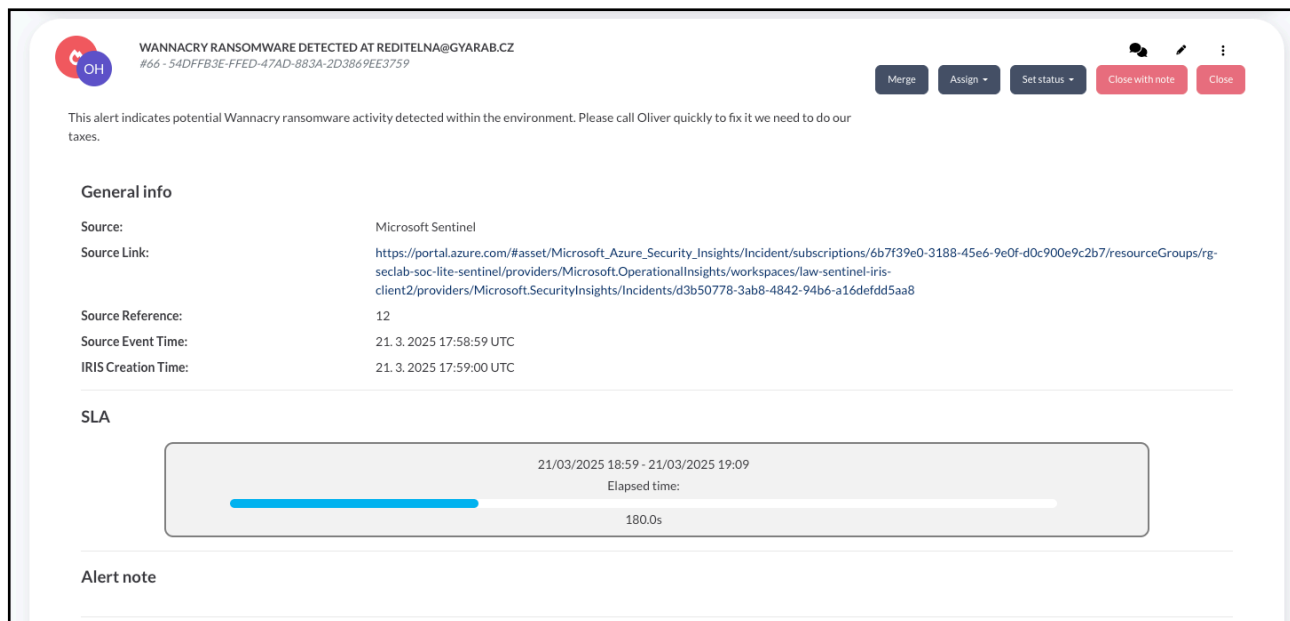
      <label class="row col-md-12" style="...">
        <progress class:completed={currstate === MyState.PAUSED} class:breached={SLABreached} max={duration} value={elapsed}></progress>
      </label>

      <div class="row col-md-12" style="...">
        <p>{elapsed.toFixed(1)}s</p>
      </div>
    </div>
  </div>


  {#if duration === elapsed}
    <p class="row col-md-12" style="..."><strong>SLA breached!</strong></p>
  {/if}
</div>
```

Obr. č. 29 - finální Progress bar




A výsledek v grafické podobě.



Obr. č. 30 - SLA v běžícím stavu



**WANNACRY RANSOMWARE DETECTED AT REDITELNA@GYRAB.CZ**  
#66 - 54DFFB3E-FFED-47AD-883A-2D3869EE3759



This alert indicates potential Wannacry ransomware activity detected within the environment. Please call Oliver quickly to fix it we need to do our taxes.

**General info**

Source:

Microsoft Sentinel

Source Link:

[https://portal.azure.com/#asset/Microsoft\\_Azure\\_Security\\_Insights/Incident/subscriptions/6b7f39e0-3188-45e6-9e0f-d0c900e9c2b7/resourceGroups/rg-seclab-soc-lite-sentinel/providers/Microsoft.Operationallnights/workspaces/law-sentinel-iris-client2/providers/Microsoft.SecurityInsights/Incidents/d3b50778-3ab8-4842-94b6-a16defdd5aa8](https://portal.azure.com/#asset/Microsoft_Azure_Security_Insights/Incident/subscriptions/6b7f39e0-3188-45e6-9e0f-d0c900e9c2b7/resourceGroups/rg-seclab-soc-lite-sentinel/providers/Microsoft.Operationallnights/workspaces/law-sentinel-iris-client2/providers/Microsoft.SecurityInsights/Incidents/d3b50778-3ab8-4842-94b6-a16defdd5aa8)

Source Reference:

12

Source Event Time:

21. 3. 2025 17:58:59 UTC

IRIS Creation Time:

21. 3. 2025 17:59:00 UTC

**SLA**


21/03/2025 18:59 - 21/03/2025 19:09

Elapsed time:




240.0s

**Alert note**

Obr. č. 31 - SLA splněn - časovač se zastaví a zezelená



**SIGN IN FROM A NEW LOCATION - P2@GYRAB.CZ**  
#70 - 6A23520F-7862-421C-9BF6-0648CE1623BC



No description provided

**General info**

Source:

Microsoft Sentinel

Source Link:

[https://portal.azure.com/#asset/Microsoft\\_Azure\\_Security\\_Insights/Incident/subscriptions/6b7f39e0-3188-45e6-9e0f-d0c900e9c2b7/resourceGroups/rg-seclab-soc-lite-sentinel/providers/Microsoft.Operationallnights/workspaces/law-sentinel-iris-client2/providers/Microsoft.SecurityInsights/Incidents/fdade498-1eda-4ea6-af73-c5b1752d761d](https://portal.azure.com/#asset/Microsoft_Azure_Security_Insights/Incident/subscriptions/6b7f39e0-3188-45e6-9e0f-d0c900e9c2b7/resourceGroups/rg-seclab-soc-lite-sentinel/providers/Microsoft.Operationallnights/workspaces/law-sentinel-iris-client2/providers/Microsoft.SecurityInsights/Incidents/fdade498-1eda-4ea6-af73-c5b1752d761d)

Source Reference:

15

Source Event Time:

22. 3. 2025 9:25:48 UTC

IRIS Creation Time:

22. 3. 2025 9:25:49 UTC

**SLA**

22/03/2025 10:25 - 22/03/2025 22:25

Elapsed time:

3600.0s

SLA breached!

Obr. č. 32 - SLA breach

## 6. IRIS v Azure a deployment

Jak nastavit prostředí a začít vyvíjet IRIS lokálně jsem popsal na githubu, budu se zde proto věnovat nastavení instance IRIS, která je přístupná z internetu. Jelikož potřebujeme vyvíjet nebo testovat integraci s Microsoft Sentinel, který běží v cloudu, musí být IRIS přístupný „z venku”.

První je potřeba vytvořit v Azure (nebo vlastně u jakéhokoliv VPS providera) virtual machine. Z mé zkušenosti by měla stačit standardní B2s. Operační systém jsem zvolil Ubuntu Server 24.04 LTS. Po vytvoření bude potřeba jít do Networking > Network settings > Create port rule > inbound port rule a povolit port 8000 pro API. Potom se přihlaste pomocí SSH.

```
sudo apt update
```

```
git clone -b develop https://github.com/Rumburaq2/iris-web.git
```

```
cd iris-web
```

```
cp .env.model .env
```

```
#odkomentovani proměnné IRIS-ADM-PASSWORD nastavíme default admin heslo
```

```
nano .env
```

```
sudo snap install docker
```

```
sudo docker compose -f docker-compose.dev.yml build
```

```
cd ui
```

```
sudo apt install npm
```

```
npm install
```

```
#postaví front-end
```

```
sudo npm run build
```

```
cd ..
```

```
sudo docker compose -f docker-compose.dev.yml up
```

IRIS by teď měl být dostupný na adrese <https://<server-ip>/>. Přihlásit se můžete pomocí username administrator a předtím v .env File nastaveným heslem.

Vyzkoušíme konektivitu k API:

```
curl -H "Authorization: Bearer <API-token>" http://<server-ip>:8000/api/ping
```

Bearer token (API token) lze najít pod přihlášeným uživatelem v IRIS v sekci My Settings.



Dále je potřeba jít do IRIS > Advanced > Customers > IrisInitialClient > Edit customer a nastavit samotné SLA. Použijeme např. Řetězec “Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday:8:20”, který definuje SLA na všechny dny v týdnu od 8 do 20 hodin.

Následně můžeme ručně poslat testovací alert.

```
curl -X POST http://<server-ip>:8000/alerts/add \
-H "Authorization: Bearer
4cOvRl5C1oleJP3u0HJLnM3GXflG1qDHAPSSYSHEEglJrPE1oRwYmj23aqwKt7cfRg1yXxEY4
vzPmtSAg36Tvg" \
-H "Content-Type: application/json" \
-d '{
  "alert_title":"Low-reputation arbitrary code executed by signed executable",
  "alert_description":"This is a test alert, courtesy of Olie",
  "alert_source":"Test Source",
  "alert_source_ref":"Test-123",
  "alert_source_link":"https://source_link.com",
  "alert_source_content":{
    "_id":"",
    "contextId":"",
    "description":"",
    "entities":[],
    "title":"",
    "URL":""
  },
  "alert_severity_id":4,
  "alert_status_id":3,
  "alert_context":{},
  "alert_source_event_time":"2024-12-28T03:00:30",
  "alert_note":"",
  "alert_tags":"",
  "alert_iocs":[
  ],
  "alert_assets":[
  ],
  "alert_customer_id":1,
  "alert_classification_id":1
}'
```

Nakonec už jen povolíme modul IrisWebnhooks a aplikujeme konfiguraci popsanou již na obrázku č. 17, str. 20. Nyní vytvoříme testovací instanci Sentinelu. Přejděte do Microsoft Sentinel > create new > create new Log Analytics Workspace. (Obr. č. 33)

Home > Microsoft Sentinel > Add Microsoft Sentinel to a workspace >

## Create Log Analytics workspace ...

**Basics** Tags Review + Create

**i** A Log Analytics workspace is the basic management unit of Azure Monitor Logs. There are specific considerations you should take when creating a new Log Analytics workspace. [Learn more](#)

With Azure Monitor Logs you can easily store, retain, and query data collected from your monitored resources in Azure and other environments for valuable insights. A Log Analytics workspace is the logical storage unit where your log data is collected and stored.

**Project details**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ Azure subscription 1

Resource group \* ⓘ oliver-vm-iris-dev\_group  
[Create new](#)

**Instance details**

Name \* ⓘ oliver-gyarab-law ✓

Region \* ⓘ West Europe

Obr. č. 33 - vytváření Log Analytic Workspace

Obdobně se proklikáme UI a vytvoříme Microsoft Sentinel.

Následně stačí přejít na <https://portal.azure.com/#create/Microsoft.Template> a použít template pro jednotlivé Logic Apps a nastavit jejich Automation rules v Sentinelu (jako popsáno na str. 23). Poté přejděte do Log Analytic Workspaces > Access control > Add role assignment a přidejte managed identitu právo Sentinel Responder. Ve výsledku máme plně funkční integraci mezi Microsoft Sentinel a portálem IRIS.

## 7. GPLv3 licence

Projekt IRIS je vydáván pod licencí GPLv3, což zaručuje mnoho svobod, ale i určitá omezení. Díky GPLv3 licenci máme právo dílo (náš fork původního projektu) modifikovat, kopírovat, dál toto dílo i jakoukoli odvozenou verzi rozšiřovat a to za úplatu či zdarma.

Naopak důležité omezení GPLv3 licence je, že i odvozené dílo (modifikovaná verze původního díla) musí vyjít pod stejnou licencí. Je tomu tak proto, aby se zachovala původní práva uživatelů a zabránilo zneužívání softwarovými společnostmi. Tento koncept se odborně nazývá copyleft<sup>9</sup>.

Jelikož je projekt IRIS pod licencí GPLv3, podléhá jí tedy i moje práce.



Obr. č. 34 - logo GPLv3 licence

---

<sup>9</sup> Autor neuveden. *GNU General Public License* [online]. [cit. 29.3.2025]. Dostupný na WWW: [https://cs.wikipedia.org/wiki/GNU\\_General\\_Public\\_License](https://cs.wikipedia.org/wiki/GNU_General_Public_License)

## 10. Závěr

Projekt splnil zadání. Práce definuje nejdůležitější pojmy vztahující se k tématu, například SOC, SLA, bezpečnostní incident, SIEM. takže i člověk neznalý v oblasti cybersecurity pochopí, co bylo cílem a jak je implementováno řešení. Práce definuje co je to a jak vypadá bezpečnostní incident. Také ukazuje, jak práce s incidentem vypadá v platformě IRIS i SIEM Microsoft Sentinel.

Následuje detailnější popis toho, proč je IRIS (centrální systém pro právu bezpečnostních incidentů) potřeba, a to zvládání několika zákazníků najednou.

Následně popisují cíl práce - obousměrná integrace (nebo-li synchronizace) incidentů mezi IRIS a Microsoft Sentinel. **Obecně bez propojení s SIEM (Security Information and Event Management) je platforma IRIS sama osobě k ničemu. Protože dosud taková integrace (mezi SIEM Microsoft Sentinel <—> IRIS) nebyla vytvořena, jedním z hlavních cílů bylo právě tuto konkrétní integraci vytvořit.**

V práci je popsán seznam všech akcí mezi Sentinel a IRIS, které musí moje řešení vykonávat. Jmenovitě přiřazení vlastníka incidentu, změna stavu incidentu (new, in progress, closed), severita (low, medium, high) i komentáře k incidentu.

Dále je popsáno, jak fungují proč byly pro implementaci vybrány Logic Apps, a jejich fungování. Práce popisuje Logic App pro synchronizaci incidentu směrem z IRIS do Microsoft Sentinel, ale i obráceně, a potřebnou konfiguraci v IRIS pro nastavení *webhooků*. Také je vysvětlen tzv. *trigger* (spouštěč) Logic Apps, což jsou automatizační pravidla v Microsoft Sentinel, a nastavení system assigned managed identity dané Logic App pro její autorizaci.

Následuje detailnější popis vnitřního fungování platformy IRIS (docker kontejnery) a popis práce nad rámec zadání - a to počítání SLA - doby řešení incidentu. Funkcionalita počítání SLA je implementována v JavaScript framework Svelte a ukázky kódu s detailnějším popisem jsou v práci také zahrnuty. Finálně prezentuji výsledky funkční obousměrné integrace i počítání SLA u jednotlivých incidentů.

Závěrem popisují, jak moje řešení nasadit v prostředí Azure - od instalace platformy IRIS na VM a její spuštění a základní nastavení, až po deploy Logic Apps. Kvalitní návrh a použití parametrizace v Logic Apps zaručuje, že je řešení lehké přenositelné a rozšiřitelné, a navíc robustní a zabezpečené. V poslední kapitole se krátce věnuji GPLv3 licenci.

Kód volně přístupný (open source) na Github - <https://github.com/gyarab/2024-4e-hurt-hauerteUnifiedPortal>), což umožňuje nejen snadné nasazení (deployment), ale zároveň mohou tento *fork* uživatelé dál rozvíjet, nebo dokonce publikovat vlastní derivaci pod licencí GPLv3.

## 11. Zdroje obrázků

Obr. č. 1 - Autor neuveden. SIEM Use Case Development [online]. [cit. 25.2.2025]. Dostupný na WWW: <https://academy.hackthebox.com/module/211/section/2253>

Obr. č. 2 - Autor neuveden. MICROSOFT.COM. *Vyšetřování incidentů s využitím služby Microsoft Sentinel* [online]. [cit. 27.3.2025]. Dostupný na WWW: <https://learn.microsoft.com/cs-cz/azure/sentinel/investigate-cases>

Obr. č. 21 - KERNEL. DFIR IRIS. *Quick Start* [online]. [cit. 27.3.2025]. Dostupný na WWW: [https://docs.dfir-iris.org/latest/getting\\_started/](https://docs.dfir-iris.org/latest/getting_started/)

Obr. č. 34 - Autor neuveden. *Licencia GPLv3*: Licencia GPLv3 [online]. [cit. 29.3.2025]. Dostupný na WWW: <https://xn--deepinenespaol-1nb.org/>