3.1/3.4 Data Analysis, Explanation and Documentation

Comecei analisando os payloads enviados ao destino e verificando se existiam parâmetros utilizados em vulnerabilidades conhecidas como XSS, SQLi, XXE, LFI, Command Execution, Dorks e possíveis arquivos de configuração expostos.

Com isso foi possível identificar que existiam diversos endereços de IPs que estava justamente efetuando não só uma varredura através de ferramentas prontas, mas também haviam payloads mais estruturados para pontos específicos.

Através destres payloads foi possível identificar diversos endereços de IPs de origem e seus respectivos dispositivos e agentes de navegação. Com isso foi possível determinar que os ataques estavam sendo originados de locais e dispositivos diferentes.

Seria possível identificar se os resultados foram OK de acordo com os bytes recebidos pela requisição, porém precisaria efetuar os payloads para ter dados de cada requisição e validar se os mesmos receberam dados.

Coletei então o top 10 IPs mais ofensores e também o top 10 payloads mais utilizados para algumas validações.

Dados coletados:

Top 10 - Payloads executados:

Command: “cat payloads.csv | cut -f12 -d";" | sort | uniq -c | sort -gr | head -10”

102 /<iframe src='javascript:alert(1)'></iframe>
102 /../../../windows/win.ini
97 "/<meta http-equiv='refresh' content='0
94 /../../../etc/passwd
94 /%00%01%02%03%04%05%06%07
93 /<marquee><img src=1 onerror=alert(1)></marquee>
93 /../../../../../../../../../../etc/shadow
91 /../../../../windows/system32/cmd.exe
90 /shell.php?cmd=cat%20/etc/passwd
87 /login.jsp?user=admin'--

Outros payloads críticos encontrados:

79 /api/v1/users?search=; DROP TABLE users

Top 10 - IPs de origem por quantidade de requisições (**IpInfo.io API**)

Command: “cat payloads.csv | cut -f1 -d"," | sort | uniq -c | sort -gr | head -10”

Qtde - IP - Cidade - País - Região - AS da empresa responsável pela range de IPs - TimeZone

156 53.153.77.110 "Sindelfingen" "DE" "Baden-Wurttemberg" "AS31399 Mercedes-Benz Group AG" "Europe/Berlin"
119 208.150.99.181 "Sunnyvale" "US" "California" "AS3561 CenturyLink Communications, LLC" "America/Los_Angeles"
116 125.227.246.131 "Tainan" "TW" "Taiwan" "AS3462 Data Communication Business Group" "Asia/Taipei"
115 185.24.37.122 "Maribor" "SI" "Maribor City Municipality" null "Europe/Ljubljana"
114 222.30.33.183 "Shanghai" "CN" "Shanghai" "AS4538 China Education and Research Network Center" "Asia/Shanghai"
114 209.158.28.49 "Reston" "US" "Virginia" "AS701 Verizon Business" "America/New_York"
114 129.53.13.62 "Bedford" "US" "Massachusetts" "AS367 DoD Network Information Center" "America/New_York"
112 21.166.16.150 "Columbus" "US" "Ohio" "AS749 DoD Network Information Center" "America/New_York"
112 192.19.249.147 "San Jose" "US" "California" null "America/Los_Angeles"
112 159.168.200.38 "Zürich" "CH" "Zurich" "AS28686 Aveniq AG" "Europe/Zurich"

Somente do **payload** 01, foi possível identificar 95 **ips diferentes** efetuando o mesmo payload, alguns IPv4 e outros IPv6.

Isso sugere que o ataque está vindo de origens diferentes.

Como não temos o HTTP-Status das requisições efetuadas, entendo que não seja possível definir se algum dos payloads obteve sucesso na requisição, sendo ela GET ou POST. Caso tivessemos os status, seria possível determinar quais payloads conseguiram efetuar um bypass da proteção existente e retornaram valores com possível vazamento de dados.

3.2. Risk Identification and Policy Development

No cenário atual, entendo que a medida mais rápida e efetiva seria a implementação de um WAF na frente da aplicação, desta forma poderíamos aplicar diversos filtros que iriam barrar estas requisições e, principalmente, diminuiriam a quantidade de requisições inválidas repassadas aos demais serviços como API e Database.

Juntamente com isso, iria identificar junto ao time de desenvolviment o esforço necessário para efetuarmos validações nos campos que estão recebendo os dados e permitindo estes payloads, porém isso pode demorar um pouco e acabar não sendo tão efetivo a curto praso. Também utilizaria medidas como SAST, SCA e DAST para avaliar a segurança da aplicação antes que ela seja publicada.

Também validaria com os times de DevOps quais são as medidas utilizadas atualmente para evitar este tipo de ataque nas aplicações e iria fazer um levantamento do cenário atual e aplicações que precisam passar pelo mesmo processo de avaliação.

O risco apresentado nestes payloads, caso explorados com sucesso, poderiam não apenas comprometer dados dos navegadores dos usuários, mas também poderiam expor dados armazenados nos bancos de dados da empresa, juntamente com um possível ataque de ramsonware em casos mais graves onde um SQLi poderia resultar em um acesso via terminal.

3.3. Implementation

Aqui como sugestão de curto prazo, seria importante aplicarmos um WAF (Web Application Firewall) na frente das aplicações críticas para que este tipo de ataque seja barrado antes mesmo de afetar o desempenho das aplicações.

Controles como RateLimit, Block por GeoLocation e bloqueios de requisições com payloads conhecidamente maliciosos (XSS,SQLi,XXE,Path Transversal,IDOR e outros)

Medidas a médio e longo prazo seria efetuar treinamentos, avaliação de código e acompanhamento das vulnerabilidades através de um processo de gestão de vulnerabilidades.

3.4. Innovation

Seria de extrema importância que fossem criadas guildas de segurança e um processo de Security Champions para que assuntos como este fossem discutidos e entendidos por todo o time de desenvolvimento e devops, desta forma, seria possível identificar lacunas nos times e treinar pessoas de acordo com a maior fragilidade do ambiente, focando então no que é crítico e prioritário para que o ambiente não seja comprometido de alguma forma.

Atividades interativas com os times de desenvolvimento, treinamentos sobre vulnerabilidades e riscos e acompanhamento frequente com as equipes para avaliar a evolução da maturidade das aplicações.

Também seria importante adicionar ferramentas de avaliação de segurança nas pipelines de modo que o código seja analisado antes mesmo de ir para o ambiente de produção, evitando assim que vulnerabilidades mais críticas sejam expostas a partir de um ambiente exposto.

Também seria interessante construírmos cenários de padronização de código, homolgação de bibliotecas e políticas de codeReview diretamente nos versionadores de código, minimizando os riscos em entregas futuras.

Script utilizado para coletar dados via IPInfo.io:

```
#!/bin/bash
for cada in $(cat ips | cut -f2 -d","); do
    qtde=$(cat ips |grep "$cada" | cut -f1 -d",")
    ipinfo=$(curl -s --location "https://ipinfo.io/$cada?token=$token")
    city=$(echo $ipinfo | jq '.city')
    country=$(echo $ipinfo | jq '.country')
    region=$(echo $ipinfo | jq '.region')
    org=$(echo $ipinfo | jq '.org')
    timezone=$(echo $ipinfo | jq '.timezone')
    echo "$qtde $cada  $city  $country  $region  $org  $timezone"
done
```