Detecting Man in the Middle Attacks with Canary Requests

Brian Wallace | Senior Security Researcher

CYLANCE™

$ whoami

- Security research, Software Engineering, learning Data Science
- Senior Security Researcher at Cylance
- Twitter: @botnet_hunter
- Lead researcher on Operation Cleaver
- Big fan of open source development
  - https://github.com/bwall/
  - https://github.com/CylanceSPEAR/
  - ssdc – ssDeep file clustering
  - bamfdetect – Static botnet configuration extraction
  - GetNETGUIDs – Extract MVID/TypeLibID from .NET Assemblies (integrated into VirusTotal)

Outline

- MITM and Attacks Leveraging MITM
- Current MITM Detections
- Changing the Game
  - Canary Requests
    - Request Modules
    - Analysis Modules
- Implementation Status
- Per tool Examples
- Future work/direction

MITM and Attacks Leveraging MITM

- A MITM state is an attacker gaining control over a victim’s connection
- Attacks leverage a MITM state take advantage of the state to attack
- A MITM state can be difficult to detect
  - Passive attacks/sniffing can leave little to no trace
  - The MITM state could be made possible because of things out of our control
- Attacks leveraging a MITM state are more plausible to detect
  - Data is modified
  - Expected behavior changes

Current MITM Detections

- Detecting attacks leveraging MITM are generally done per application or connection
- Tend to rely on the software to ensure the connection is secure
  - HTTPS/SSH validate with protections built into SSL/TLS
- Some cases require the user to verify data was received properly
  - Checking the hash of a download
- Responses to a MITM
  - At best, the application reacts
  - More common, connection just fails

Changing the Game

- MITM is a system level attack in most cases
- Detection and response should happen on system level (as well)
- Should have dedicated application checking for indicators of MITM
- Act as another level of protection on top of the application/connection level checks
- Leading strategy is to make “Canary Requests”

Canary Requests

- Train the Canary Request
  - Make a request a few times from a trusted network (Request Module)
  - Analyze responses (Analysis Modules)
  - Identify consistencies and inconsistencies
- Testing/Checking
  - Make the same request (Request Module)
  - Analyze and compare response to training responses (Analysis Modules)
  - Identify if the inconsistencies are different than those from training
  - If different, alert the user
- If user considers the differences benign, added as a trusted response

Canary Requests – Request Modules

- Request Modules implement configurable network requests
- Additionally parse responses
- Example: HTTP request module
  - Makes GET request to configured URL
  - Allows definition of HTTP headers
  - Parses the response into status code, headers, remote IP, and content
- Parsed response information is passed to the Analysis Modules with trusted responses

Canary Requests – Analysis Modules

- Compare current request and previously gathered request
- Each module focuses on small data point
  - Allows the analysis comparisons to identify what is relevant
  - Simpler to implement
- Example: HTTP Status code comparison
- Example: HTTP Compression comparison
- Each module relevant to a request returns a brief analysis

Implementation Status

- All Python 2.7
- Developed in Kivy
  - Allows for single Python code base to be deployed cross platform
    - Windows/OSX/Linux
    - Android/iOS too!
- Service Component
  - Does the canary requests
  - Continuously runs
- UI Component
  - Alerts only

Per Tool Examples - MITMf

- MITMf
- https://github.com/byt3bl33d3r/MITMf
- Man in the Middle Framework
- Implements wide variety of attacks, passive and active
- By default, converts all HTTPS URLs in HTML content to HTTP
- Detected by HTTP content comparison
- https://youtu.be/fDbQMk5OMZw

Per Tool Examples – Zarp + MITMProxy

- Zarp for getting MITM state
- https://github.com/hatRiot/zarp
- MITMProxy to intercept/analyze traffic
- https://mitmproxy.org/
- MITMProxy feature allows HTTP compression stripping (intended to be transparent)
- HTTP Request module with Accept-Encoding: gzip
- HTTP Compression Analysis module identifies the sudden lack of expected compression
- https://youtu.be/vEPU3FICqEw

Per Tool Examples - Responder

- Responder
- https://github.com/SpiderLabs/Responder
- Responds to LLMNR/NBT-NS/mDNS requests to control connections
- mDNS Request Module
- Local/Remote/Empty/Comparison IP Analysis modules
- Analysis modules identify sudden change in resolution of mDNS response
  - Not actually expecting a response
  - Response is internal, expected external
  - Response is a different IP than expected
- https://youtu.be/d8oWPesBFUY

Future Work/Direction

- More request and analysis modules
- Improved user interface
- Change UI communication method
- Utility interface (proxy support, on demand testing)
- User configurable whitelisting
- Active learning to handle false positive mitigation
- Automated system level responses
- Make versions available to all support platforms

Any Questions?

- Twitter
  - @botnet_hunter
  - @CylanceSPEAR
- https://github.com/CylanceSPEAR/mitmcanary
- https://github.com/bwall
- https://blog.cylance.com/