

Planting C4

Cross-Compatible External C2 for All Your Implants

Scott Taylor

Disclaimer

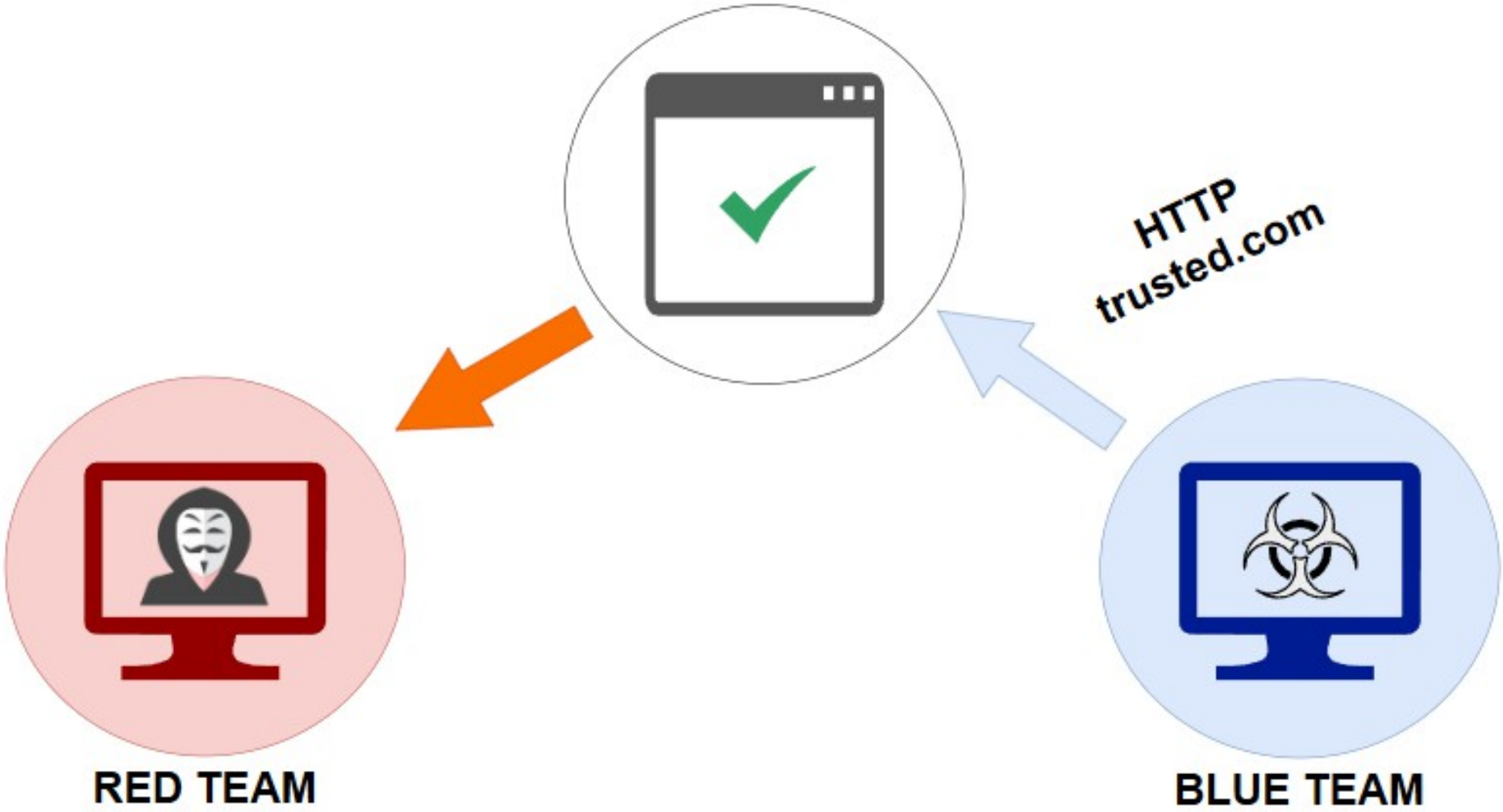
My remarks are in a personal capacity representing my own views, opinions, and experiences. My statements do not reflect the views, positions, or activities of any Sony Group company.

whoami

- Red Team Operator for numerous organizations
  - Sony (current)
  - T. Rowe Price
  - MITRE
- Started as a Linux Sysadmin
- OSCP, Red Hat, Cisco, CompTIA
- B.S. Information Technology
- M.S. Cybersecurity Technology



External C2



LOLC2 (@mthcht)



Telegram
CounterStrike 1.6
Cloudflare
Strava
Google Translate
Microsoft Teams
Discord
Splunk
Lichess
Mattermost
Mastodon
Whatsapp
Microsoft Printer
Microsoft Azure Application Proxy
Asana
Google Sheet
Microsoft Sharepoint
Google Drive
Google Calendar
Google Slides
X
Onedrive
Microsoft Tasks
Jira
Twitter
Github
Youtube
Pastebin
Reddit
Dropbox
Gmail
Slack
Cisco Webex
DuckDuckGO
Microsoft Graph
Instagram
Zoom
Virustotal
Zulip
Notion
Microsoft Outlook
Microsoft Power Automate
Microsoft Azure Functions
Soundcloud
Spotify
Matrix
Openai
Claude
Microsoft Azure Blob Storage

In the Wild



- T1102: Web Service
  - Technique added May 31st, 2017
- Used in the wild by threat actors
- Leverage trusted, legitimate sites

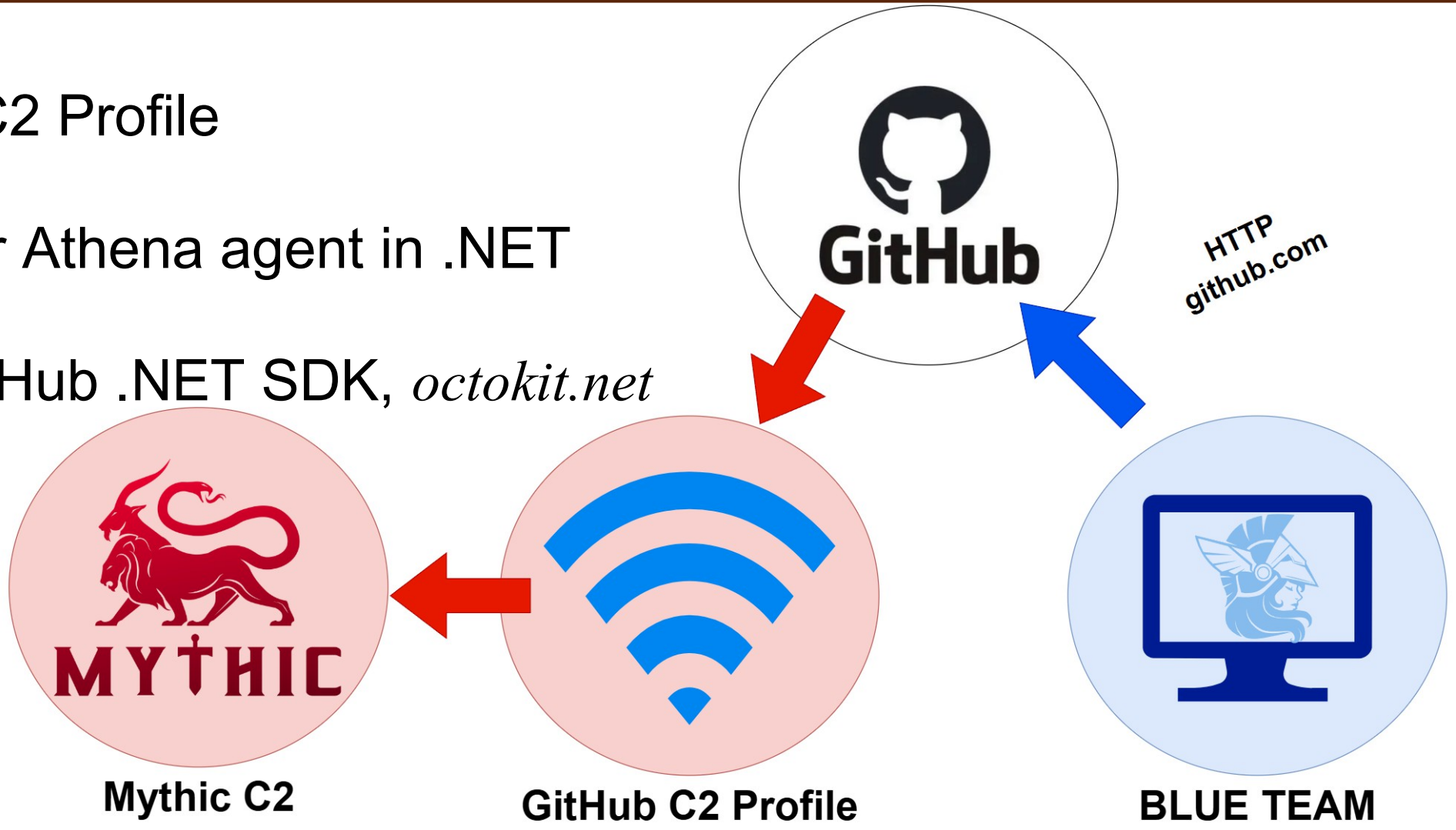
ID	Name	Description
G0050	APT32	APT32 has used Dropbox, Amazon S3, and Google Drive to host malicious downloads.[2]
C0040	APT41 DUST	APT41 DUST used compromised Google Workspace accounts for command and control.[3]
G1044	APT42	APT42 has used various links, such as links with typo-squatted domains, links to Dropbox files and links to fake Google sites, in spearphishing operations.[4][5][6]
S1081	BADHATCH	BADHATCH can be utilized to abuse `sslip.io`, a free IP to domain mapping service, as part of actor-controlled C2 channels.[7]
S0534	Bazar	Bazar downloads have been hosted on Google Docs.[8][9]
S0635	BoomBox	BoomBox can download files from Dropbox using a hardcoded access token.[10]
S1063	Brute Ratel C4	Brute Ratel C4 can use legitimate websites for external C2 channels including Slack, Discord, and MS Teams.[11]
S1039	Bumblebee	Bumblebee has been downloaded to victim's machines from OneDrive.[12]
C0017	C0017	During C0017, APT41 used the Cloudflare services for C2 communications.[13]
C0027	C0027	During C0027, Scattered Spider downloaded tools from sites including file.io, GitHub, and paste.ee.[14]
S0335	Carbon	Carbon can use Pastebin to receive C2 commands.[15]
S0674	CharmPower	CharmPower can download additional modules from actor-controlled Amazon S3 buckets.[16]
S1149	CHIMNEYSWEEP	CHIMNEYSWEEP has the ability to use use Telegram channels to return a list of commands to be executed, to download additional payloads, or to create a reverse shell.[17]

Why are these external C2 options not available in every C2 framework?



GitHub C2

- Mythic GitHub C2 Profile
- Client written for Athena agent in .NET
- Used official GitHub .NET SDK, *octokit.net*

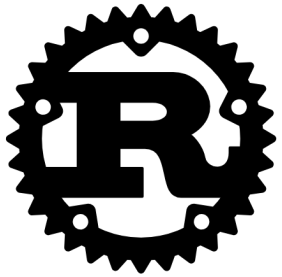


Mythic C2 Agents

Agent	Author	Language
Poseidon	its-a-feature	GoLang
Athena	checkymander	.NET Core
Apollo	djhohnstein	.NET Framework
Medusa	ajpc500	Python
Thanatos	MEhrn00	Rust
Venus	mattreduce	JavaScript (VS Code Extension)

https://github.com/MythicAgents













WebAssembly

WebAssembly



- Compiled low-level binary format
- Near-native speed
- Cross-platform & architecture-independent
- Memory-safe sandboxed execution
- Supported by all major browsers
- Supported by numerous WASM runtimes?



WebAssembly System Interface (WASI)



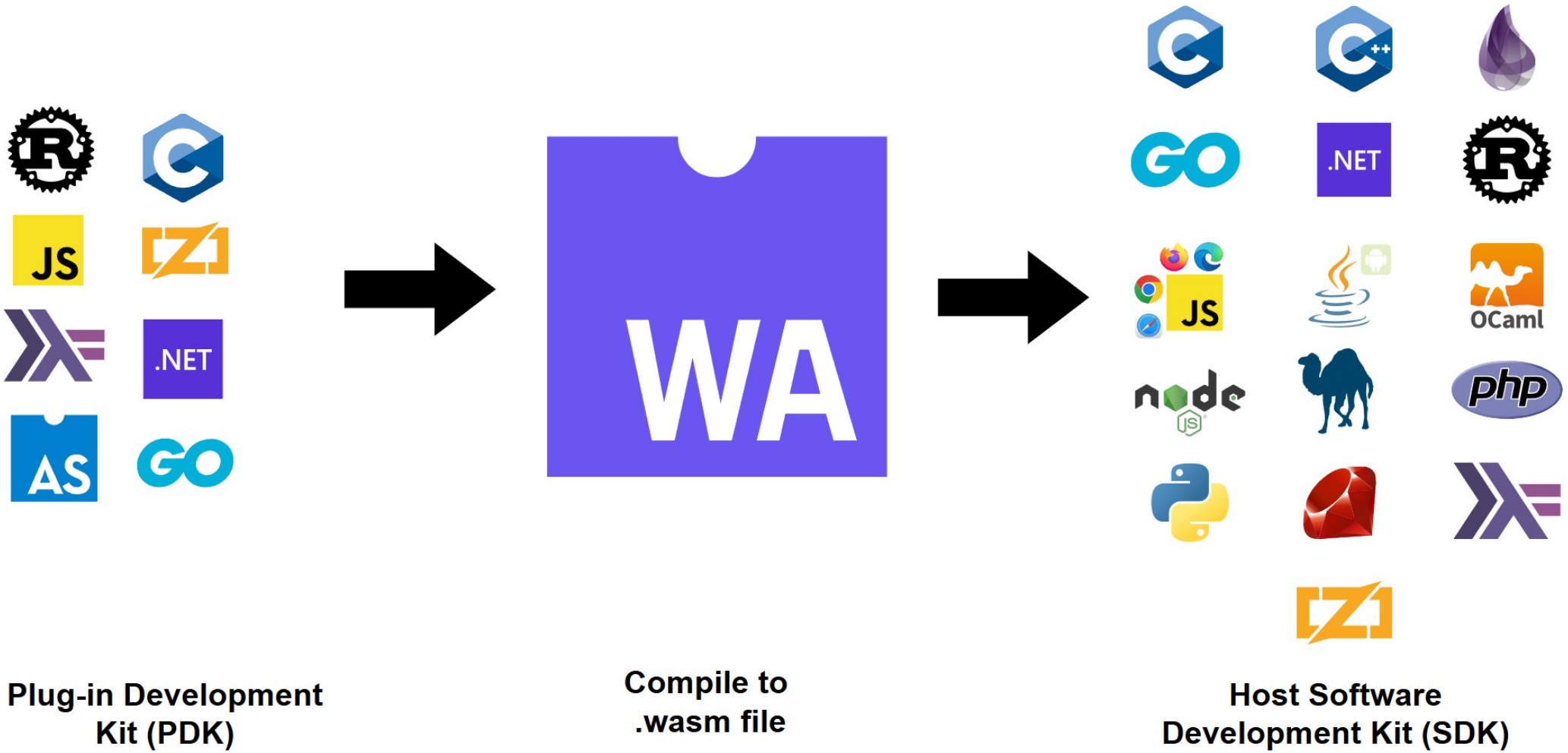
- Bridge to the real system!
- Run apps anywhere with a wasm runtime
  - Cloud
  - Server-side apps
  - Command-line tools
  - Embedded Systems
  - Plugin Systems

Extism (by Dylibso)

- Universal WASM plugin system
- Numerous language support
- Host (SDK) and Plugin (PDK) Support
- Rich WASM runtime enhancements
  - Persistent memory
  - Host-controlled HTTP
  - Execution limits, timeouts, and resource control



Extism


JS
.NET
AS
GO
WA
GO
.NET
JS
OCaml
node
php
Plug-in Development Kit (PDK)
Compile to .wasm file
Host Software Development Kit (SDK)



C4: Cross-Compatible Command and Control

Hello World



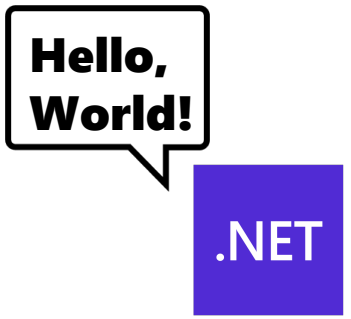
plugins > hello-world > src > lib.rs > ...

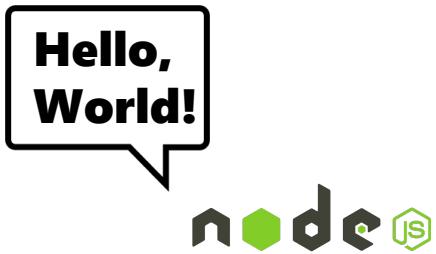
```rust
use extism_pdk::*;

#[plugin_fn]
pub fn c4() -> FnResult<String> {
    Ok(format!("Hello, World!"))
}
```









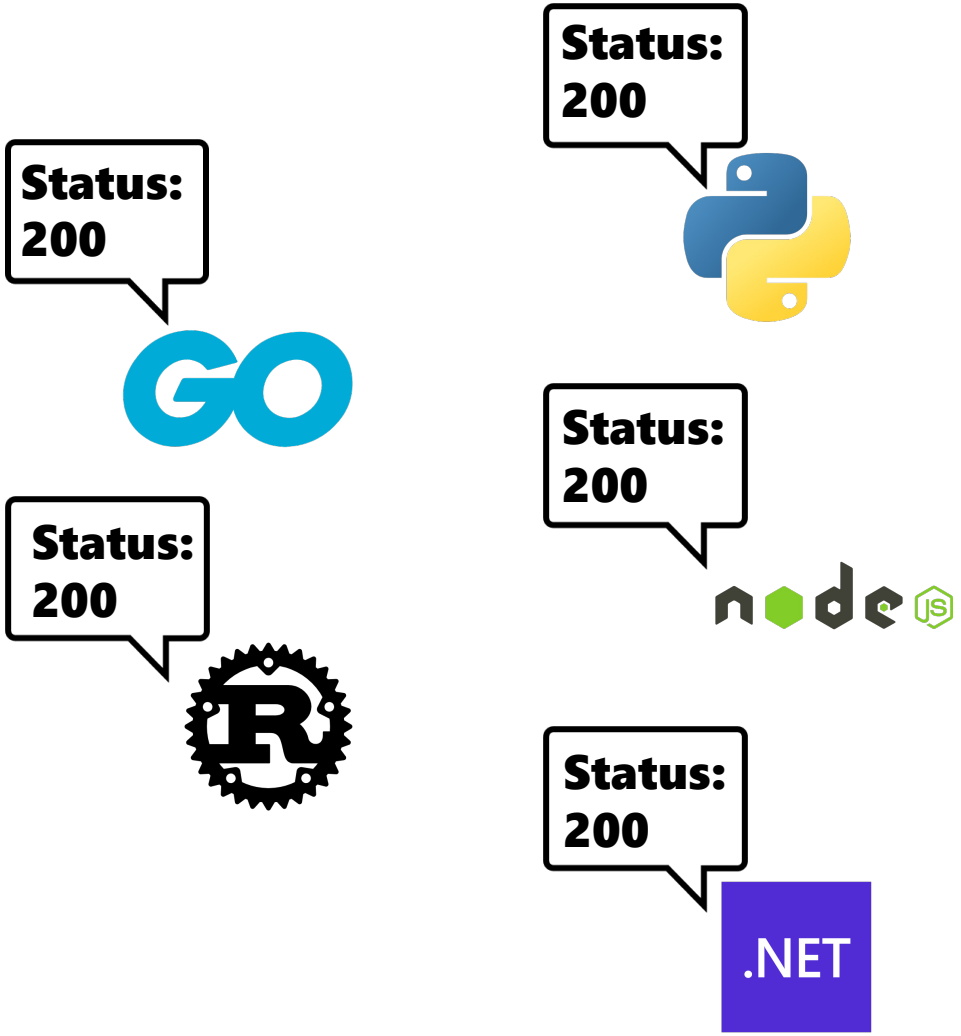


HTTP Test



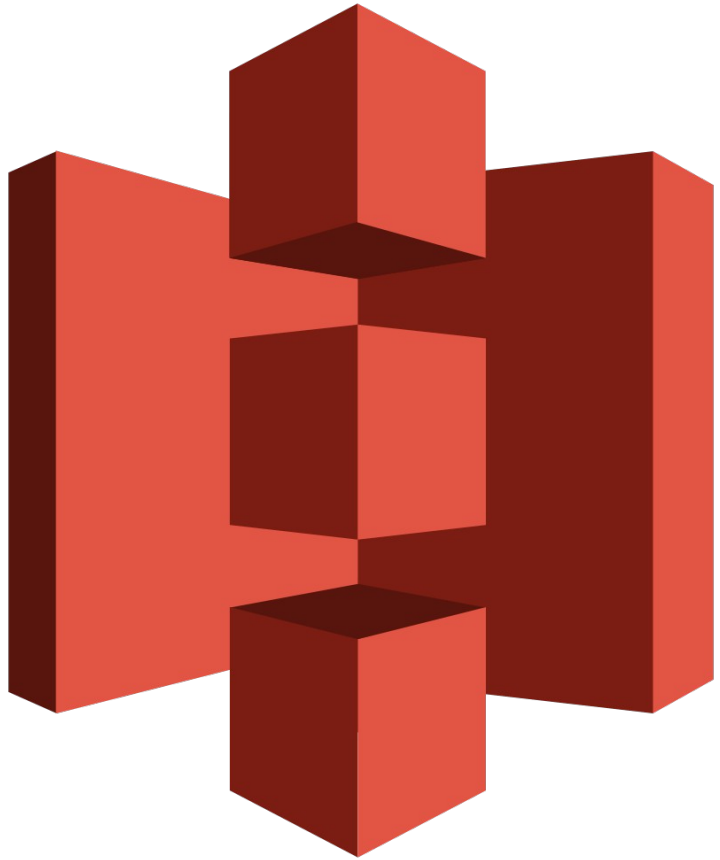
plugins > http-test > src > lib.rs > ...

```rust
use extism_pdk::*;
use serde::{Deserialize, Serialize};

#[plugin_fn]
pub fn c4(url: String) -> FnResult<String> {
    // Create HTTP request
    let req = HttpRequest::new(&url);

    match http::request::<()>(&req, None) {
        Ok(response) => {
            // Return just the status code as a string
            Ok(response.status_code().to_string())
        }
        Err(e) => {
            // Return error information
            let error = ErrorResponse {
                error: format!("HTTP request failed: {}", e),
            };
            Ok(serde_json::to_string(&error)?)
        }
    }
}
```



AWS S3

- Two actions: “send” and “receive”
  - Send – upload file with message
  - Receive – check for messages and delete after retrieval
- Messages/Files uploaded by folder of “agent id”
- File name is timestamp to avoid duplicate name



Allows for large messages

AWS S3: API Reference



- Send

```
{
    "action": "send",
    "params": {
        "agent_id": "string",
        "message": "string",
        "access_key": "string",
        "secret_key": "string",
        "region": "string",
        "bucket": "string",
    }
}
```

- Receive

```
{
    "action": “receive",
    "params": {
        "agent_id": "string",
        "access_key": "string",
        "secret_key": "string",
        "region": "string",
        "bucket": "string",
    }
}
```

Python Host: Load Plugin



examples > python > main.py > ...

```python
import extism

def load_plugin():
    with open("c4.wasm", "rb") as f:
        wasm_bytes = f.read()
    manifest = {
        "wasm": [{"data": wasm_bytes}],
        "allowed_hosts": ["*"],
    }
    plugin = extism.Plugin(
        manifest,
        wasi=True,
    )
    return plugin
```

- Import *extism* package
- Create *manifest* that allows outbound HTTP requests
- Enable *WASI*
- Return loaded plugin

Python Host: Run plugin

examples > python > main.py > ...

```python
def main():
    plugin = load_plugin()
    input_data = '''
    {
        "action":"send",
        "params": {
            "agent_id": "agent-007",
            "message": "test msg",
            "access_key": "AKIA**********",
            "secret_key": "SECRET**********",
            "region": "us-east-1",
            "bucket": "c4-testing"
        }
    }
    '''
    result = plugin.call("c4", input_data.encode('utf-8'))
    print(result.decode())
```

- Load plugin
- Setup message
- Call *c4* function and pass input
- Rinse & Repeat

Conclusion

- External C2 doesn’t have to be challenging
- WASM is continuously developing
- Start using C4 plugins today!
- Reach out @scottctaylor12

