

Feb 27, 19 8:51

agent.rs

Page 1/3

```

1  /*
2  *
3  *
4  * This code works now. it needs some serious cleanup however. wow is it gross.
5  *
6  * learn how to build and embed into shipped python:
7  *     https://github.com/mckaymatt/cookiecutter-pypackage-rust-cross-platform-publish
8  *     https://pypi.python.org/pypi/setuptools-rust
9  *     https://github.com/getsentry/milksnake
10 *     https://github.com/getsentry/libsourcemap/blob/master/setup.py
11 *
12 * echo '{"agent_token":"super secret", "master_key": "secret", "timeout":30, "port":"6278"}' |
13 * agent
14 *
15 * and the request:
16 *curl --data '{"agent_token":"super secret"}' --header "Content-Type: application/json" http://localhost:6278
17 *
18 */
19 //#[macro_use]
20 extern crate serde;
21 #[macro_use]
22 extern crate serde_derive;
23
24 extern crate serde_json;
25 // #[macro_use]
26 extern crate bitwarden;
27 extern crate daemonize;
28 extern crate secstr;
29 extern crate tiny_http;
30
31 // use rouille::Request;
32 // use rouille::Response;
33 // use std::collections::HashMap;
34 use std::io;
35 // use std::sync::Mutex;
36 use secstr::*;
37 // use std::time::{SystemTime, UNIX_EPOCH};
38 use std::time::{Duration, Instant};
39 use std::fs::File;
40 use daemonize::Daemonize;
41
42 // This struct contains the data that we store on the server about each client.
43 #[derive(Debug, Clone)]
44 struct SessionData {
45     login: String,
46 }
47
48 #[derive(Deserialize)]
49 struct Setup {
50     master_key: String,
51     agent_token: String,
52     timeout: u64,
53     port: isize,
54     agent_location: String,
55 }
56 /*
57 struct Secrets {
58     agent_token: SecStr,
59     master_key: SecStr,
60 }
61 */
62 #[derive(Deserialize)]
63 struct TokenRequest {
64     agent_token: String,
65     exit: bool,
66 }
67 #[derive(Serialize)]
68 struct TokenResponse {
69     master_key: String,
70     error: String,
71 }
72 //
73 // echo '{"agent_token":"super secret", "master_key": "secret", "timeout":30, "port":"6278"}' |
74 // target/debug/agent
75
76 fn serve(setup: Setup) {
77     let secret_agent_token = SecStr::from(setup.agent_token);
78     let secret_master_key = setup.master_key;
79     // let secrets = Secrets {
80     //     agent_token: SecStr::from(setup.agent_token),
81     //     master_key: SecStr::from(setup.master_key),
82     // };
83     // set timeout to 1 year if bitwarden doesn't give us one.
84     let mut timeout = Duration::new(31536000, 0);
85     if setup.timeout > 0 {
86         timeout = Duration::new(setup.timeout, 0);

```

Feb 27, 19 8:51

agent.rs

Page 2/3

```

87     }
88     let port = setup.port;
89     let server = tiny_http::Server::http(format!("localhost:{}", port)).unwrap();
90     println!("Now listening on localhost:{}", port);
91
92     let start_time = Instant::now();
93     let mut exit = true;
94     let receive_timeout = Duration::new(1, 0);
95
96     // loop for freaking ever, or agent_timeout happens..
97     while exit {
98         // block for up to 1 second.
99         let request = match server.recv_timeout(receive_timeout) {
100             Ok(rq) => rq,
101             Err(e) => {
102                 println!("error: {}", e);
103                 break;
104             }
105         };
106         if request.is_some() {
107             let mut rq = request.unwrap();
108             let url = rq.url().to_string();
109             let method = rq.method().to_string();
110             let mut content_type = String::from("");
111             for header in rq.headers() {
112                 println!("header: {} : {}", header.field, header.value);
113                 if header.field.to_string() == "Content-Type" {
114                     content_type = header.value.to_string();
115                 }
116             }
117             println!(
118                 "remote request from:{} requesting: {} method:{} content_type:{} ",
119                 rq.remote_addr().to_string(),
120                 url,
121                 method,
122                 content_type
123             );
124             // assert_eq(request.remote_addr(), "127.0.0.1");
125             if method == "POST" {
126                 println!("post!");
127                 if content_type == "application/json" {
128                     println!("JSON!");
129                     let mut content = String::new();
130                     rq.as_reader().read_to_string(&mut content).unwrap();
131                     let token_request: TokenRequest = serde_json::from_str(&content).unwrap
132
133                     if SecStr::from(token_request.agent_token) == secret_agent_token {
134                         if token_request.exit == true {
135                             println!("exit requested");
136                             std::process::exit(0);
137                         }
138                         // let local_master_key = secret_master_key;
139                         let token_response = TokenResponse {
140                             error: "".to_string(),
141                             master_key: secret_master_key.clone().to_string(),
142                         };
143                         let response = tiny_http::Response::from_string(
144                             serde_json::to_string(&token_response).unwrap(),
145                         );
146                         // return Response::json(&token_response);
147                         let _ = rq.respond(response);
148                     } else {
149                         let token_response = TokenResponse {
150                             error: "Invalid agent_token.".to_string(),
151                             master_key: "".to_string(),
152                         };
153                         let response = tiny_http::Response::from_string(
154                             serde_json::to_string(&token_response).unwrap(),
155                         );
156                         let _ = rq.respond(response);
157                     }
158                 } else {
159                     let token_response = TokenResponse {
160                         error: "Must send Content-Type: application/json".to_string(),
161                         master_key: "".to_string(),
162                     };
163                     let response = tiny_http::Response::from_string(
164                         serde_json::to_string(&token_response).unwrap(),
165                     );
166                     let _ = rq.respond(response);
167                 }
168             }
169             // end server handler
170
171     }
172     let now = Instant::now();

```

Feb 27, 19 8:51

agent.rs

Page 3/3

```

172         if now.duration_since(start_time) > timeout {
173             exit = false;
174         }
175     }
176     println!("exiting because of timeout hit.");
177 }
178 }
179
180 fn main() {
181     let mut input = String::new();
182     io::stdin().read_line(&mut input).unwrap();
183     let setup: Setup = serde_json::from_str(&input).unwrap();
184     println!(
185         "setup: {:?} {:?} {:?}",
186         setup.master_key, setup.agent_token, setup.timeout
187     );
188     let stdout = File::create(format!("{}/agent.out", setup.agent_location)).unwrap();
189     let stderr = File::create(format!("{}/agent.err", setup.agent_location)).unwrap();
190     let daemonize = Daemonize::new()
191     .pid_file(format!("{}/agent.pid", setup.agent_location)) // Every method except 'new' and 'start'
192     .chown_pid_file(true) // is optional, see 'Daemonize' documentation
193     .working_directory(format!("{}", setup.agent_location)) // for default behaviour.
194     .user("nobody")
195     .group("daemon") // Group name
196     .group(2) // or group id.
197     .umask(0o027) // Set umask, '0o027' by default.
198     .stdout(stdout) // Redirect stdout to '/tmp/daemon.out'.
199     .stderr(stderr); // Redirect stderr to '/tmp/daemon.err'.
200     .privileged_action(|| "Executed before drop privileges");
201
202     match daemonize.start() {
203         Ok(_) => println!("Success, daemonized"),
204         Err(e) => eprintln!("Error, {}", e),
205     }
206     serve(setup);
207 }

```