Censorship circumvention tools face an arms race as they work to evade increasingly motivated censors. Tools which have distinctive features can be detected and blocked by censors (e.g., Tor is actively targeted by censors around the world). As a result, there is increasing interest in disguising censorship circumvention traffic as benign protocols. SkypeMorph [1] and StegoTorus [2] are two pluggable transports [3] for Tor which aim to mask Tor traffic as Skype traffic, and a combination of Skype, HTTP and Ventrilo, respectively.

While these pluggable transports are able to capture features of the traffic they aim to imitate (e.g., inter-packet timings, packet size distributions), Houmansadr et al. [4] point out that imitating a protocol is not enough to evade detection – i.e., if circumvention endpoints do not run the application they aim to hide within, they are vulnerable to censors who may perform active probing to observe that the endpoint does not actually implement the full functionality of the mimicked application. As such, Houmansadr et al. advocate that circumvention schemes should actually run the application they are using as cover, rather than mimicking properties of the cover traffic.

Towards this goal, FreeWave uses the Skype application as a modem to transmit IP data between two endpoints [5]. However, Geddes et al. demonstrate that even running the cover application is not enough to avoid detection – i.e., if circumvention endpoints do not run the application they aim to hide within, they are vulnerable to censors who may perform active probing to observe that the application they are using as cover, rather than mimicking properties of the cover traffic.

The Castle approach: In order to create a covert channel mechanism that is general to the majority of games in the real-time strategy genre, Castle exploits two key properties.

- Most real-time strategy games share a common set of actions. Specifically, the ability to select buildings and assign a location where units created/trained in a building should go. This location is called a "rally point", and we denote the command of setting the rally point for units created in a given building by \texttt{SET-RALLY-POINT}.
- Most real-time strategy games provide a replay option which saves every players' moves to disk (for later playback). Therefore, all in-game commands are written to disk where they can be read and decoded in real-time, with little effort.

Castle consists of two main components to send and receive data. These are illustrated in Figure 1. Sending is done by encoding data into game commands and then executing them via desktop automation. The receiving process monitors the log of game commands and decodes this list to retrieve data sent via the system.

Figure 2 overviews how the Castle system could be used to relay data from outside of a censored region to a client within the region. The client first installs Castle (e.g., as a browser extension). The Castle client then initiates a game through a game lobby (or directly with the client outside of the censoring region). The client
Fig. 1: Overview of data flow for sending and receiving in Castle. Shaded components are implemented as part of Castle while the others use existing off-the-shelf software.

Fig. 2: Overview of how Castle can be used as a proxy for clients within censoring countries.

in the censoring region can then encode and send data (e.g., Web requests) as game moves that can be decoded by the client outside of the censoring region. The game client outside of the censoring region can then act as a proxy to retrieve censored content and send it via Castle to the client in the censoring region.

We demonstrate the feasibility of our approach by prototyping on two different games (one open-source, and one extremely popular closed-source real-time strategy game) with minimal development overhead (< 9 hours for development and deployment) and show its resilience to a network adversary. Specifically, our results show that Castle is:

- **Extensible**: Castle’s strength comes from its pluggable architecture which allows it to be easily ported to any number of games. As an example, it took a bright undergrad less than 6 hours to complete a basic port (e.g., player personality, strategies employed, scenario type, map, number of players, etc.), flow level features may vary widely between game instances. Castle (under all configurations) generated traffic was well within the variance seen in real human vs. human game traffic, for both – inter-packet times and packet sizes.

- **Secure**: Castle is resistant to attacks such as IP/port filtering and deep-packet inspection since it actually executes the game application. Further, we find that Castle is also resistant to attacks that depend on analysis of flow level features such as packet sizes and inter-packet times. Since the traffic generated by a standard multiplayer game is strongly dependent on many parameters (e.g., player personality, strategies employed, scenario type, number of players, etc.), flow level features may vary widely between game instances. Castle (under all configurations) generated traffic was well within the variance seen in real human vs. human game traffic, for both – inter-packet times and packet sizes.

**Castle Source Code:** [https://github.com/bridgar/Castle-Covert-Channel](https://github.com/bridgar/Castle-Covert-Channel)

**Video demonstration:** [https://www.youtube.com/watch?v=zpjZJuvMhDE](https://www.youtube.com/watch?v=zpjZJuvMhDE)

**REFERENCES**


