# CreepyDOL

Brendan O'Connor

Malice Afterthought, Inc.

June 30, 2013

CreepyDOL, the Creepy Digital Object Locator, is a distributed tracking system that uses low-cost hardware sensors, a robust communications system, and simple observation to give near-real-time identification of humans and tracking capabilities to anyone.

## Architecture

CreepyDOL has several components that work together to provide the service.

### F-BOMB

The F-BOMB (Falling/Ballistically-launched Object that Makes Backdoors) is the sensor node platform for CreepyDOL. F-BOMB's original incarnation was presented at ShmooCon 2012.[1] F-BOMB was designed to provide an extremely-low-cost sensor platform that was small and light enough to be deployed by UAV, reasonably powerful and power-efficient, and crucially, to be as low-cost as possible, to allow anyone to experiment with low-power distributed computing for whatever

[1] http://www.youtube.com/watch?v=Vm_cHb8Mm9w

their target purpose might be. The F-BOMB v1 used a Marvell Sheeva-based board; v2, used in CreepyDOL, instead uses a Raspberry Pi, which is quite a bit smaller in its reference board, more customizable, runs a wider variety of software, and is somewhat cheaper.[2] Added to this are two USB WiFi chips (Ralink RT5370, which is one of the only chips I found that supports monitor mode, Master (AP) mode, and injection while being physically small; if space was not an issue, the TP-Link WN series of USB WiFi adapters, using the Atheros chipset, would also be a good alternative for slightly more money), a USB hub which also provides power to the Raspberry Pi, an SD card to serve as non-volatile storage, a USB power adapter, and a case. Total cost per node, when buying 10 nodes worth of parts at once: $57.08, including shipping. A price and part list is attached at the end of this document.

The F-BOMB runs Debian Linux, using the standard Raspberry Pi distribution.

## Reticle

Reticle provides a command, control, and communications ($C^3$) platform for very-low- cost computers to receive tasking, exchange data, and change missions on the fly, designed especially for ad-hoc deployments of one-time-use computers. Reticle was developed with the assistance of the DARPA Cyber Fast Track program in 2012, and presented at Security B-Sides Las Vegas in 2012.[3]

---

[2] While the Sheeva development board cost $99, the F-BOMB instead used PogoPlug devices, which while they listed for $150, were regularly being sold for $25-$35. The Raspberry Pi Model A, used in CreepyDOL, costs $25 list.

[3] http://www.youtube.com/watch?v=Csl8tzb2Big

Reticle is a system with many moving parts. Each node runs CouchDB, which is the primary storage system. In front of CouchDB is an Nginx installation, which is used to verify client-side SSL certificates (due to CouchDB Bug 1448,[4] while CouchDB theoretically supports client SSL certificates for replication, and while CouchDB can indeed use a client certificate while connecting to another node, it crashes if it is asked to *verify* a client SSL certificate; therefore, CouchDB is configured to use a client certificate when a node is a client, but not to verify a client certificate when it is a server, and Nginx handles verification in that case). Nodes use CouchDB's "pull" replication to gather commands and data from all other nodes; since each Reticle node will attempt to continuously gather all information from all nodes known to it, the effect is to create a "gossip" or "contagion" protocol to move data. This means that data collected on one node will quickly be exfiltrated to the rest of the network, and commands will move in the same way.

Nodes don't bring a backhaul Internet connection with them. Instead, they search for and connect to local wireless signals (continuously, to allow connections when moving; since CouchDB's replication is both efficient and asymmetric, a continuous connection is not needed). Nodes communicate with each other via Tor, to prevent anyone local to the network from determining with whom the nodes are communicating. Per-node SSL certificates mean that a node can be cut off by simply revoking its certificate, if there is a suspected compromise. Finally, nodes use "grenade-style" encryption: the disk where all the Reticle data and configuration resides is an encrypted drive, and the key is inserted on a removable USB drive at

[4] https://issues.apache.org/jira/browse/COUCHDB-1448

boot, from which it is read and stored only in RAM, before the USB drive is removed. This "pull pin, throw device toward privacy insurance claimant" encryption methodology strikes a balance between security and usability, in that those users deploying the node do not need to know how to use the encryption, what the encryption key is, or even type anything into the device.

## NOM Filters

The Reticle system can run any task on its hardware, so we configure it to sniff for local wireless signals, then run them through what we call NOM Filters: for Nosiness, Observation, and Mining. These are per-application filters that look for data that they understand, then extract any useful data from it; for instance, one filter looks for Dropbox traffic, another looks for online dating traffic, and a third looks for web-based email. Each pulls out anything it can recognize, and throws it in the Reticle database. (That can be anything from just the fact that a user *uses* Dropbox, since most of its traffic is encrypted, to a dating profile picture, for other filters, to a user's email address.) Then, rather than shipping all the traffic across Reticle, only this filtered information is pushed into the database. Crucially, we don't just need "traffic" signals; even the standard WiFi beacons, where a device looks for friendly networks (as used in Jasager[5]) give us a great deal of information, as well as allowing us to track nodes even when they don't connect to any area networks.

Reticle is a leaderless C3 system; no node is more special than others. One node, however, is used by CreepyDOL as the "data sink;" it takes in data gleaned

[5] http://www.digininja.org/jasager/

from other sources, then issues delete commands to remove the data from the (limited) node storage devices. Those deletes will propagate across the network just as the data did, along with any new commands (for instance, updated filters.) The data saved by the sink node is saved into the backend, for additional NOM filtering that can't be done on individual nodes; for instance, noting a series of times and places at which a user was present, and developing paths and pattern analysis for the user, or noticing that two or three devices always show up together, and thus are likely attached to the same user (for instance, a cell phone and a tablet). All of these conclusions are, again, thrown into the backend.

## Visualization

The CreepyDOL visualization system was developed under a second DARPA Cyber Fast Track contract, and modified for CreepyDOL. It uses Shark, a Hadoop Hive-compatible cluster working on top of the Spark distributed system, to process large amounts of data (terabytes+) on a cluster---including on Amazon EC2.

To visualize the data, a new visualization tool was written in Unity, the 3D gaming engine. The visualization can show the devices and their location on a 3D map of the area in question, along with data about the user---from their photo, to their email addresses, to where they usually go for coffee (when the pattern analysis reaches such a conclusion).

To make the visualization program lightweight, all queries are offloaded onto the Shark cluster; this means that rather than having to deal with terabytes of data in local memory, a visualization user can just request updates over the network from the cluster. This means that the visualization program can run on simple

hardware; a Mac laptop or an iPad is perfectly sufficient, and the program should run on Android, Windows, or Linux as well (if recompiled; the Unity engine can output to all of these). To translate between the Shark cluster and the visualization program, we use a simple shim server, written in Sinatra (Ruby).

To show the map of the area in question, we use OpenStreetMap-style "slippy map" tiles, that allow the program to quickly understand an area's map; since we use this known web standard, we can quickly substitute in other tiles. (Tiles are served from a separate HTTP server, co-located with the Sinatra shim layer.) In addition, when we have surveillance footage or the like, the visualization program can overlay the video anywhere on the map (again, specified by the OSM "slippy map" coordinates), in conjunction with a time display, to allow users to correlate video with network events (for instance, to look at video of who enters a store when the locator says a target is entering).

## Test Methodology

We deployed CreepyDOL nodes to several different locations in a populous, well-travelled, section of Madison, WI. **To prevent badness, we programmed the NOM system to look only for traffic from devices we owned; no "random stranger" data was collected at any time.** With that constraint, we were able to capture a significant amount of useful data about the devices, including photographs of their owners, correlation between devices owned by the same person, and some "this is where he hangs out"-type data.

## Future Work

The next step is to expand CreepyDOL; it's easy to expand the number of filters running both on the nodes and on the backend, which increases the types of traffic CreepyDOL can recognize. Beyond that, opening CreepyDOL up as an open-source project (which will be done on August 1, 2013) will allow people to use CreepyDOL to solve a large variety of problems, as well as to explore just how much data their organization leaks on a daily basis through simple, inadvertent WiFi traffic.