All.Net Analyst Report and Newsletter

Welcome to our Analyst Report and Newsletter

Simple Stupid Secure Short-term Servers

In the late 1990s, we developed White Glove Linux (Garrett Gee and me). It was a small bootable CD for a Linux version with some special features. Minimalist in various ways, able to run from RAM without any disk required in the hardware setup, configurable with a floppy which you could write lock for a read only bootup, restart from power up or reset button in about 30 seconds on a regular PC circa the late 1990s. (http://all.net/WG/index.html)

The idea at that time was to very quickly setup a system to do whatever you want, so we built in lots of tools, help, etc. Secure in the sense of read-only with no unnecessary processes, interfaces, etc. It only talks to the world if you let it, sets up for promiscuous silent Ethernets for covert sniffing, supports special applications like the deception wall, Responder, and so forth.

Times have changed

The one big problem for using it today is that you need a fixed IP address in order to use it as a server. And if you have a fixed IP address, folks can try again and again, and do distributed denial of services attacks, and on and on. Of course you can proxy to it wherever it is, use reverse tunnels and so forth, but what we find useful today is the extremely short-term specialized server for special functions like moving files about and having a more secure chat.

How do you bootstrap a reasonably secure chat?

Suppose the bad guys are out to get you, and they are serious, well funded, with expertise. You want to have an ability to have short online conversations relatively free from concern about the bad guys knowing what was said. So how do you do it?

- **You get a clean system:** Go purchase a Chromebook for cash and create a new ID.
  - Or better yet – get a really cheap PC and run Linux on it (from USB bootup).
- **You get a safe place to sit:** Try any place that has free Internet access.
  - A different place each time, seating with your back against the wall.
- **You use a fixed point in IP space to have the conversation**
  - You need some where to go on the Internet to have the conversation
- **The fixed point has to be very temporary**
  - When the conversation ends, the fixed point has to go away

So you do all that, except how do you setup a temporary fixed point?

I have an account at Amazon Web Services (free tier), and you can have one there and at lots of other similar cloud services. These services provide automated means to startup a standard server (linux various version), and tear them back down. They get fixed external IP addresses (which can change with time unless you pay for them to be longer term). You create one, boot it, make it do the things you want, then destroy it when done.

Putting it together

In about 8 minutes, if you practice just a bit, you can create a server with a fixed IP address, load up some minimal software (openSSL, stunnel, a trivial server for example). For a quick setup chat system, load and configure ‘shellinabox’, ‘screen’, and a few login IDs that type into one screen area which appends to a file and tail -f a common file in the other area. On the White Glove CD, this was called SeeSay, but in that case it ran over ssh, which most users will find too hard to get running compared to the Web browser approach of shellinabox.

I have it all scripted out (for copy and paste into the remote server) and it takes about 8 minutes from start to operable chat session with one or more remote partners. After the chat, it takes less than minute to terminate the machine and have it returned to the AWS resource pool. I also use the same basic process to get a version of ‘woof’ (file upload/download) running over SSL (using stunnel – self-signed certificates).

I could do that much faster by automating it

So why don’t I? Because if I set up an image or a Kubernetes automated mechanism, the bad guys could gain access to my AWS, Microsoft, or whatever controls and alter the image to give them copies. Automation supports subversion of the automation. Of course the service provider could try to do some such thing, but the costs of doing this by preserving or watching activities on every server they pop up and down would be prohibitive, and I can create a new account pretty quickly (with a cash card purchased with cash, …). It’s worth the 8 minutes for the extra protection… in some cases.

How ‘secure’ is it?

The term ‘secure’ is a complicated one, so I will simplify it. This approach provides some real advantages over trusting a service, like dropbox for files or signal, and so forth for chat:

- The providers can watch what goes on and change the way it works, including setting it up to track you as you use the system over time.
  - You are playing in a small pool for any one of these services, whereas the totality of the cloud ecosystem is available for the approach here. The bad guys would have to track a lot more stuff in a lot more places and would run out of resources if they tried to do this in a broad sense. Each instance looks like every other instance from afar. Scanning for a server popping up on AWS, then identifying it as one of ours (using the same technology as millions of other servers) is just the start of gaining access. Finding the user IDs and then the memory area storing the chat, oops… it shut down… Not impossible, but pretty tough for the level of effort we put in.
- It’s not as good as running it in RAM like White Glove does
  - Sort of… but on the other hand, the cloud providers’ storage arrays that get overwritten with new instances all the time (unless you configure them to back up, etc.) are probably not very persistent in terms of retaining the few Kbytes of a conversation of few Megabytes of a file transfer. An estimate from a few years ago was that AWS supports 1-2 zettabytes of storage. That’s a trillion gigabytes, about $2^{70\text{-}71}$ bytes… 1-2,000,000,000,000,000,000,000 bytes. And the things we pop up will likely get overwritten from residual disk storage (if it is ever actually stored on disk as opposed to cached) pretty soon… And even if it persists, finding it ...

- But the SSL certificates are “insecure”
  - Actually, they are self-signed – which is not the same as insecure by a long way. The Web browsers tell us they are insecure, etc. but they also do not depend on 3rd parties for their security – and of course major providers can be subverted for the signature services, and have been in the past. They could be forged and intercepted by a machine-in-the-middle attack, but to do this, you need to know where the endpoints are. One endpoint is as one coffee shop or hotel lobby, as is the other. And the endpoint at AWS can be anywhere across their global infrastructure. It only takes a few seconds to start the SSL session, and once it is started, machine-in-the-middle won’t work for that session. And we can watch the sessions on the server (and have all parties see them as they come and go) and do an instant shutdown if anything looks suspicious (any user can simply click to kill it).
- But…
  - There is a long list of buts that can be lit upon. And nothing is perfect in this world. But as you think of them, consider how much effort it would take to implement them compared to the time and effort required to put these mechanisms in place. Watching keystroke timing from network traffic? Detecting differences in keystroke sounds with a parabolic antenna? Listening to the mumbling of the person typing as they say things to themselves? Watching the screen reflections off the walls? How many more? Planting a covert camera in the target’s clothing? Go for it!
- How about the endpoints themselves?
  - This is, of course, always a weak point in any communications system. If you can get to the computers on the ends and subvert them, you can win. That’s why it’s best to buy with cash from any old computer big box store, not register it but subvert it to load Linux (from a USB without network enabled) from the start, and only connect it to networks for these sessions. But getting an off-the-shelf chromebook is probably almost as good…

Didn’t I forget something?

No doubt, I forgot lots of things… but one in particular remains. Like any such system, there is an initialization problem. How do I get the URL to the other participant? If the bad guys get it first…

- This is always an issue, but it goes like this. Key distribution without prior shared secrets always remains vulnerable as it depends at some point on original identification. If you don’t know who you are talking to, you cannot be sure it is who you think it is.
- On the other hand, if I merely tell you on an open channel the URL for access, you can use it. If someone else ALSO uses it, that is easily detected by the number of users. If it’s a case of trying for a machine in the middle attack, they had better be very quick about it. They will have perhaps 30-60 seconds to get the infrastructure in place.

Conclusions

Simple Stupid Secure Short-term Servers (SSSS) have long been a nice way to get a few bits of private communication done with nominal effort and reasonable protection. Enjoy...

