Pidgin News

Subscribe via RSS 2.0 or Atom!

Junk security reporting for cash — and credit?

June 25, 2015 03:14 PM by Ethan Blanton

A few times a year, Pidgin gets a rash of junk security reports from "independent security researchers". Typically we get 3–5 or more reports in the span of just a couple of days, reporting "vulnerabilities" such as HTTP POST submission of plain-text passwords to TLS-encrypted GNU Mailman subscription preferences login. (As many of you will know, by default Mailman will *email your password* to you every month. In addition, password retrieval is as simple as entering a subscribed email address and waiting for the password to arrive in that address's inbox.) The report typically also asks for consideration in the form of a bounty or finder's fee.

The typical defining characteristics of such an email are:

1. The security flaw being reported is trivial-to-irrelevant.
2. The mechanism by which the flaw operates is often protocol- or configuration-related, such as the presence or absence of certain HTTP headers or the method used for a form submission.
3. The report does not indicate that the "researcher" understands the flaw or even personally verified its existence; often one gets the feeling from this and the previous point that it was identified by an automated tool.
4. The reporter shows no understanding of the entity being contacted, often (for example) referring to the Pidgin project as a "company" or suggesting that it is a large and wealthy organization.
5. The reporter asks for some sort of reward.
6. *All of the reports in a given burst come from the same general locale, often India.*

The first three points on this list make the reports seldom useful and somewhat of a nuisance. The third point, in particular, means that when the reporter receives a polite reply to the effect of "we know, it's not a problem and here's why," they often persist in repeating textbook explanations of the flaw with no reference to the *effective implications* of the problem (or lack thereof). Sometimes this degenerates into threats of exploitation or disclosure to the highest bidder, though not often.

The fourth and fifth points conflict somewhat with the sixth point in trying to understand why this happens and attempting to address the problem. Points 4 & 5 suggest that the reporter's primary objective is financial gain, and coupled with 1–3 that they recently found out about this issue (or issues like it) and saw an opportunity to make a quick buck off large organizations with mediocre security skills. This would lead me to believe that the rashes of back-to-back reports are prompted by, perhaps, media coverage of bug bounties or reporting of a new-and-interesting configuration error in large numbers of sites. Unfortunately, the error or flaw being reported is often not new or interesting.

I believe that the sixth point is most instructive, and I think there's something the community can do about it. I suspect that these rashes of reports are in fact *course assignments* for some sort of course on computer security being taught at an academic institution (and probably various courses at various institutions). I suspect that students are instructed to learn about some common vulnerabilities, find an example on the web, and contact the site administrators with their findings.

I cannot object to the general principle behind such an assignment. However, I *can* object to the way it is being specifically handled. If my suspicions are correct, then I believe that it is the responsibility of the course instructor to vet the students' individual reports for relevance and correctness before sending them on to the affected organizations. I would also leave out the bounty-seeking in an educational setting, but I recognize that requesting a bounty for valid flaws is an accepted practice in the commercial world, and that identifying valid flaws is a valuable service.

An alternate, but related, possibility is that some security course is discussing vulnerabilities and disclosures, and students are taking it upon themselves to contact various organizations with flaws they found using automated tools to try to make some cash on the side. I find this less likely given the very short time frames in which such reporting bursts occur (often just a few days).

In either of these education-related cases, there's something that can be done about the problem. If you are an instructor teaching such a course, please emphasize *responsible reporting*, which includes understanding both the mechanism and the impact of the flaw, and determining whether it is likely to be relevant to the affected organization. In addition, as mentioned above, screening reports before they are sent out would be the neighborly thing to do.

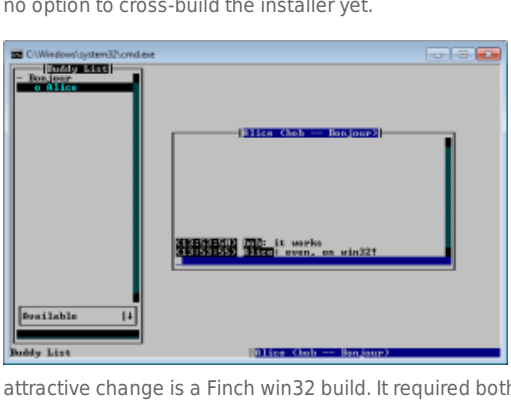
In the case of an organization like Pidgin, we take vulnerability reports seriously and expend some effort validating their existence and/or relevance. As we are an all-volunteer organization with a limited amount of time at our disposal, dealing with bogus and junk reports waste valuable resources. Report responsibly!

More Windows, more features

May 13, 2014 04:29 PM by Tomasz Wasilczyk

With a great help from LRN, who sent initial set of patches for fixing autotools-based build for win32 and assisted in the work, I have finally managed to make 3.0.0 cross compilation possible and easy. It also involved fixing minor win32 problems and enabling features that were not accessible for this platform before.

The most important change is making linux-to-windows cross-compilation almost as easy as normal build. The whole effort (not counting installation of mingw* packages) is reduced to setting two environmental variables (`PKG_CONFIG` and `PKG_CONFIG_PATH`) and adding a single switch to the configure script: `./configure --host=i686-w64-mingw32 && make`. That's all! For now, there is no option to cross-build the installer yet.



On the other hand, the most end-user-attractive change is a Finch win32 build. It required both libgnt and Finch fixes, which made those two quite usable on this system.

Moreover, I finally implemented all remaining features required for using OTR plugin with Finch. There is still one missing – you cannot browse key list yet, but it's not crucial for a daily use. This change is not related to Windows and there may be minor problems when running the new OTR plugin on win32, but I will face all of them by the chance of integrating it with Pidgin tree.

An example of less significant, but still useful Windows related change is GTK3/gstreamer-1.0 compatibility, which was an easy to achieve with the fixed autotools build. I also removed the Bonjour SDK dependency from the win32 build, as there were license issues with it – for now, you can build Bonjour prpl without it.

The last feature may be found useless by some of you, while some might like it – that's why it's optional. You can enable Filesystem Hierarchy Standard directory structure with a single configure switch: `--with-win32-dirs=fhs`. For now, it's the easiest way to prepare a working Windows build, since there is no cross-built installer yet.

Few steps towards a stable release

Pidgin 3 is not API/ABI compatible with Pidgin 2, and is only partially configuration-compatible. While the first incompatibility is necessary to move forward, the second might be really frustrating for users. Because of ABI incompatibility, libpurple2 plugins won't work with libpurple3 – their authors will eventually convert them for the new version. Configuration incompatibility may lead to loosing your data – preferences, contacts etc.

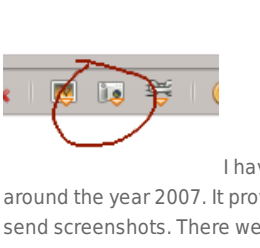
Pidgin 3 was almost backward-compatible, which allowed to switch to it flawlessly. On the other hand, Pidgin 2 was not forward-compatible. For example, it dropped all encrypted passwords, that were set with Pidgin 3. Now, it won't be able to read those (since keyring support is not implemented for 2.x.y branch), but at least it leaves them in place. I also fixed more forward-incompatibilities: handling for GTalk and Facebook XMPP accounts created with 3.0.0 version and internationalization issues related to default group ("Buddies") naming. All of these will be released in 2.x.y branch, so you will be able to switch from 3.0.0 to 2.10.10 and back without loosing your data.

There were also minor, but annoying issues fixed. Nick colors for chat participants in XMPP MUC or irc should not suffer from a low-contrast issue. I have finally made spell checking usable, by implementing a language selection sub menu for WebKitGTK. It still has some flaws, that I will work on some day: the biggest one is that the selected language is global, not per-conversation.

To make development branch more stable, I decided to focus on Coverity bug reports. Since we are allowed to maintain just one branch at once, I decided to fix all 2.x.y reports before switching to 3.0.0. For now, I fixed almost all of them (and merged fixes to the 3.0.0 branch), so it should be a bit more stable. I also updated all win32 dependencies, which should also improve stability.

Don't get me wrong, there is still a long way to the stable Pidgin 3.0.0. But it's already usable right now.

Show me your desktop



I have switched from *Konnekt* (a local Polish instant messenger) to Pidgin around the year 2007. It provided a feature absent in Pidgin, that I missed very much: an easy way to send screenshots. There were a plugin for Pidgin 2.x.y, but I didn't liked it and it wasn't working with Pidgin 3.

Ultimately, I decided to invest two days of my life and create my own plugin. It *just works*, covering all attributes I liked in other instant messengers with such feature. It's simple, stable and fits into the Pidgin UI well.

I plan to extend its functionality with another plugin. For now, it's only possible to send screenshots over protocols with inline images support. Thus, you can not use it with XMPP or IRC. The second plugin will allow for uploading images to imgur (or similar services) for protocols that doesn't support images.

Smile!

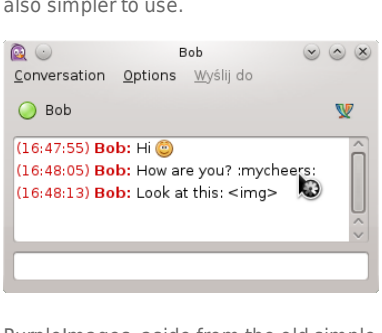
April 21, 2014 03:37 PM by Tomasz Wasilczyk

I have put a lot of effort into a feature that I don't even use: graphical emoticons. As usual, the small issue resulted in a large code refactoring. I had the task of fixing regression bugs from Pidgin 2.x.y, so I took care of broken remote smileys. It's a pleasant feature, that allows defining the list of custom smileys to use in outgoing messages. If we send one in our message and the remote client doesn't have it, the image will be automatically transferred. For protocols not supporting this feature, the bare textual representation will be shown instead of the picture.

The code responsible for emoticon handling was horrid. In 2.x.y branch it was already quite messy, but became even worse after replacing GtkIMHtml with WebKitGTK for displaying a conversation. The problem was, the old parsing code was optimised to be used with GtkIMHtml. It was totally incompatible with WebKitGTK, so the GSoC student who did a conversion made his own provisional smiley parser. It just added insult to injury.

Inline images

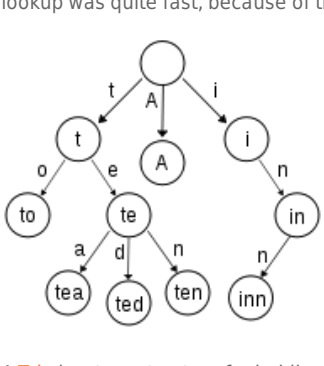
Both GtkIMHtml and WebKitGTK use purple's "stored images" to provide data for emoticon smiley. It's a generic container to hold raw image file contents, without any decoding. Its main purpose was pretty simple: to reference this data by a single integer number. Thus, an image may be embedded in HTML with `<img src="purple-image:5">`. It's quite convenient, but the API was a bit messy. Not *that* bad, but we had plans to convert it to a GObject. Instead of duplicating the exact behavior of the old implementation, I took the usual way: I wrote entirely new PurpleImage class, much richer, but also simpler to use.



PurpleImages, aside from the old simple referencing feature, have several new ones. Remote images support is the one which improves end-user experience the most. It allows defining an empty image and providing a data for it later. Now, if we receive a message with an inline image embedded (not necessarily a smiley), it will be displayed without it and the image will be shown when ready – just like its done in web browsers. It heavily impacts protocol plugins: with the old API, they had to queue messages and wait for images being transferred. Now, it's fully handled by libpurple. By the chance of deploying it, I fixed inline images support in every protocol that had it.

Tries

As I mentioned before, the smiley parser for 3.0.0 branch was provisional and the old one was not suitable for WebKitGTK. GtkIMHtml (a Pidgin 2.x.y component, despite its Gtk namespace) exposed its internal parsing mechanism and allowed to process every literal in parsed HTML. Thereby, a libpurple routine was called on every word and replaced them into an image, if hit the emoticon. The lookup was quite fast, because of trie-like implementation of GtkSmileyTree.



A Trie is a tree structure for holding multiple strings and searching them by prefixes. The primary idea of trie is that two strings with the common prefix will share the part of tree for the common segment and branch out the remaining parts. It's only a fundamental part of structure, which may be used for completely different purposes. In fact, it was used both in the old GtkSmileyTree and my new smiley parser implementation. Except for this single similarity, these two are barely related.

Instead of writing fat-but-fast smiley parser, I decided to implement independent PurpleTrie class – a trie-based text search algorithm. It allows for searching multiple strings in multiple source texts in a linear time! To be precise, it needs $O(m)$ time for building a trie (where m is the total length of provided patterns) and $O(n)$ time for searching (where n is the length of source text). The frequency of patterns in a text doesn't affect this value. The best thing in that structure is, it may also be used in libpurple plugins.

For a contrast, the interim smiley parser implementation was not focused on performance at all. Its search time could be estimated as $O(n * m * t)$, where n is the number of smileys, m is their length (assuming smileys are equal-length – in the real-world case the formula would be more complicated) and t is the text length. It's that bad, mainly because every supported (not necessarily inserted into a text) smiley is parsed separately. The $m * t$ part depends on `strstr` implementation, but for the better ones we could find another similarly bad instance of smileys configuration. With such bad complexity, it could even be exploited for a denial-of-service attack.

What's next?

In 3.0.0 branch there were a lot of smiley- and image-related issues and I haven't described all of them. But I hope I fixed all of them. You can look for details in our hg, in smiley-related commits between 302a7cb4c1ab and d598e7950c34. Now, I am focused on the permanent issue – the Windows version.

In Support of Instant Messaging Communications Freedom

February 16, 2014 02:40 AM by Ethan Blanton

As recent news events have driven home time after time, secure communications is a difficult yet important aspect of modern life. What "secure" may mean differs from person to person and from topic to topic, but certainly the typical person is somewhat uncomfortable with the idea that anyone — and in particular, large corporations or the government — might be eavesdropping on their casual communications. There are exceptions, and there are people who believe that only criminals should have something to hide, but for the moment let's assume that some measure of secrecy in private communications is warranted and/or desirable. This post is a commentary on some of the technologies that make it possible (or difficult or impossible, in many cases) to achieve secure communications on an instant messaging (IM) network.

I'm going to play a little bit fast and loose with terminology to help keep it understandable, but I also want it to be accurate. In particular, when I say that communication is *secure*, I mean *private*, specifically. For the purposes of this post, that effectively means encrypted and using some sort of authentication protocol to ensure that it's encrypted to the right person. The details are out of scope. Please contact me if you notice any particular discrepancies or incorrect statements, either arising from loose terminology or other errors.

First, where I'm coming from: I believe that essentially all communications should be secured, and I believe that that security should be very strong. By "very strong," I mean that it should be effectively impossible for any third party to eavesdrop without the acquiescence of a party participating in the communication. I have a fair amount of history in computer communications. I am a long time developer on the Pidgin instant messaging client and related software. I am on the board of directors for Instant Messaging Freedom, Inc., a non-profit organization "whose goal is to support free instant messaging software." Instant messaging is important to me as a communication technology filling the gap between real-time spoken communication and e-mail.

Types of IM security

There are a number of ways in which an IM conversation can be "secured", and not all of them have the same properties. First, there is using a *secure connection*, such as SSL or TLS between your computer and your IM service's computer. Second, there is *end-to-end security*, such as Off-the-Record Messaging (OTR). Then there are additional, more complicated options that we won't discuss — but they basically break down to a combination of one or the other of the above, or one of the above with a different remote endpoint. (\textit{E.g.,} end-to-end security between you and a remote user's IM server to query her current status.)

Secure IM connections

A secure connection to the server protects your connection from eavesdropping on the local network and the path between you and your IM service (provided that it's done correctly), but it does not protect your conversation from the IM service itself and it does not tell you anything about whether the user you're chatting with is using a secure connection. Despite these weaknesses, a secure connection to the server is *critically important*, because it protects a large amount of private information. Things like buddy lists, status updates (online, offline, away, *etc.*) from your buddies, and your own status updates pass through this connection and are not necessarily targeted to any specific other user, which makes end-to-end encryption difficult.

This is the limit of security offered by *most* commercial IM services. There are a few exceptions, like AIM's encrypted IM, but often they have a similar effective security model — for example, the only proof that the encrypted channel you're talking on is actually terminated at the remote user (and not the IM service's servers) is a certificate signed by ... the IM service itself. (The details of why this is not sufficient are outside the scope of this post. Maybe I'll write some more on that later.)

End-to-end security

End-to-end security protects your communication from eavesdropping *all the way to the other user*. If some person or organization wants to read your messages, they have to do so at your computer or your interlocutor's computer. There are a small number of protocols that support this directly, but mostly not in a particularly useful way (see the discussion of AIM encrypted IM above).[1]

The usual solution for end-to-end encryption is a third-party protocol carried on top of the IM session. There are several such protocols, but by far the most popular is the previously-mentioned Off-the-Record. OTR provides end-to-end encryption for two-party conversations with authentication of the remote user and a variety of desirable encryption protocol characteristics. (It also provides *repudiability* for situations where that may be important.)

What end-to-end security *cannot* provide is protection for all of the stuff that's intended for a very broad audience.[2] Such data (things like away messages, online status, and buddy lists) is typically[3] handled by the service's servers on behalf of its users, in such a way that necessitates more-or-less trusting the servers with the information. (This is a reason not to put sensitive information in your status message!)

Secure connections with end-to-end security

Given the individual limitations of these two forms of data security, we arrive at the inescapable conclusion that, for instant messaging services, they are *complementary* rather than redundant or simply unrelated. Secure connections to the servers provide best-effort protection for group chats, administrative information, and metadata, while end-to-end security provides strong protection for conversation content.

The pros and cons of federation

Traditional IM networks are monolithic, walled gardens — if you want to chat with a user on the network, you get an account with the single service provider for that network. There have been limited exceptions to this over the years (*e.g.*, MSN Live Messenger and Yahoo! Messenger offer(ed) some degree of interoperability), but for the most part, not only have networks been incompatible, there have been sole providers within those networks.

This kind of structure means that, while your metadata and administrative information are only ever managed by a single entity (the monolithic service provider), that service provider also sees *all* of the related metadata *etc.*, and it's necessary some giant faceless corporation. That corporation stores your buddy lists, knows when you talk with whom, knows when you're away or idle, and all kinds of other behavioral information. Moreover, it has you in a lock — if you want to talk to your buddies, you have to use its services.

The alternative to the monolithic single-provider network is a *federated service*. In a federated service, multiple (possibly unrelated) service providers cooperate in a network tied together by a common protocol, allowing users of different service providers to interact with each other without worrying about whose service is provided by whom.

In a federated structure, you still have to trust *your* service provider with all of your metadata and administrative information, but you don't necessarily have to trust any of the *other* service providers in the network. In fact, in the general case, most of them don't even know you exist! Some portion of your data will necessarily be shared with the servers your friends and interlocutors are associated with, but you can scope that sharing to a greater or lesser degree. On the other side of the coin, you also have to place at least a small amount of trust in third-party providers with whom you have no specific relationship, if you want to talk to people on those providers' servers.

The quintessential federated network is XMPP, previously known as Jabber. XMPP is a widely federated network, wherein *anyone* can run an IM server and become a service provider for other users. Conversations between users work a lot like email; if I want to chat with you, I send a message to my server, it forwards it to your server, which forwards it to you. The return path is the same in reverse. Not only can anyone put up an XMPP server, but the protocol is *entirely open and well-documented*, so there are literally dozens of server software implementations and thousands of providers already in the network. Those providers range from large, commercial entities (such as Google's Google Talk, now known as Hangouts) to tiny servers serving just one user.

The huge benefit of federation is the freedom to *choose your service provider*. Moreover, to *change* that service provider. With an open federated network like XMPP, you can even be your *own* service provider if you so desire. That's a kind of communications freedom that no monolithic provider can ever provide, by definition.

It is my opinion that the federated structure is a superior solution, security, privacy, and freedom-wise, to old-style monolithic IM networks. The majority of your sensitive data (administrative information, complete buddy list, *etc.*) is kept and managed by only one entity, and is parceled out to third-party entities only as required to provide the services you specifically request. In the specific case of a closed group of users (such as a corporate or organizational server), it may be contained entirely. I am also a strong advocate of open standards which federated solutions tend to require (to make federation possible), and which XMPP certainly provides. The ability to pick up your data and move it to another service provider with limited (or nonexistent) loss of functionality is an extremely powerful argument for the freedom of a federated solution.

Today's best practices

The foregoing basically points to a simple best practices recommendation for IM freedom and security: use XMPP, find a server you trust, ensure that you're using TLS encryption, and employ an end-to-end security solution like OTR when it matters. (Unfortunately, without complete penetration of end-to-end security solutions, "when it matters" is the best we can do. Even then it can be hard to achieve!) Today, with the availability of a number of large XMPP service providers with federation and open registration (Google, DuckDuckGo, Jabber.org, or dozens of others), as well as many fine XMPP clients (including, of course Pidgin, finch, and Adium), getting an XMPP account and finding your friends is relatively painless. Many of them will already be available if you simply add gmail address as a buddy in your XMPP client.

Secure connections are not yet provided by all XMPP servers. Among servers that do provide secure connections to clients, not all provide secure connections to other servers. (If you recall the email-like communication model of XMPP, this means that your communicatiosn with users on other servers would not be secured between servers, even assuming you trust both servers.) The excellent xmpp.net directory of public XMPP servers provides ratings for client-to-server and server-to-server communications security; look for A-rated servers. If you set up your own server, xmpp.net also provides a security validator that can be used to ensure your personal server is up to snuff.

Improving the situation

Going forward, there are a number of efforts underway to further improve the already rather good connection security situation in the XMPP network. Notably, the XMPP manifesto is (documentation of) an effort to transition the entire federated XMPP network to secured connections by May 19, 2014.

The end-to-end security situation is still a little underdeveloped, in my opinion. OTR is great, but its protocol-independent nature leaves it with a level of integration that isn't as complete as it could be. I have some early-draft notes on desirable features for a new XMPP end-to-end encryption protocol, but a lot of work remains to be done on the topic — and much of it by people with stronger crypto chops than I have.

What you can do now

The takeaway from all of this? Ditch your AIM, MSN Live Messenger, Yahoo! Messenger, or whatever other IM services you're currently using, and get on board with a federated XMPP provider. Follow the recommendations in best practices, above. Use a public server or install your own, but do it sooner rather than later. Make sure you're using TLS (or SSL, if you have to) to protect your connections, and consider installing OTR.

Then, when you're done with that, start bringing your friends over. Talk to them about the benefits of freedom in IM services, describe the insecurity of communication on traditional commercial IM services, simply tell them you're not dealing with a closed IM service any more, or whatever. Point them at this article, if you think it will help. XMPP already has critical mass, it's simply a matter of expanding the borders.

If you have the necessary background, consider contacting me about working on an integrated end-to-end encryption and authentication solution for XMPP. Join the devel@conference.pidgin.im XMPP MUC and indicate your interest.

Footnotes

[1] SILC is an example of a service that provides native secure connections *and* native end-to-end encryption. Unfortunately, it is no longer a maintained codebase, it is not well-supported by IM clients (though libpurple, and thus Pidgin does support it), and it has a problematic federation model.

[2] "Cannot" is a bit strong here, but I believe that, given the current state of the art of encryption protocols and mechanisms, it's true enough to use for the moment. Secure multiparty broadcasts with hundreds of recipients would be *very* expensive using standard techniques. The literature may (and probably will) have some answers to this problem in the future, but for now, I'll say cannot.

[3] I say typically here because I know of no non-local-network service that handles this any other way. Local network messaging (like Apple's Bonjour) has other solutions to this problem. Generally, however, you send your status updates to the server and it distributes them (possibly by way of other servers, in a federated protocol) to interested parties on your behalf.

Off-the-Record Messaging – the true privacy

October 10, 2013 03:09 PM by Tomasz Wasilczyk

Modern instant messengers claims about security and privacy. And users trusts them. But companies that delivers such services sometimes fails protecting their users privacy. The point is, that user doesn't have any control over confidentiality of his private conversations – the service provider does it.

In most cases, this schema works pretty good: user A sends a private message via an encrypted connection to a service provider and the provider sends it to the user B (also using an encrypted connection). The problem occurs, when the intermediary company reveals such message – due to software failure or dishonest employee.

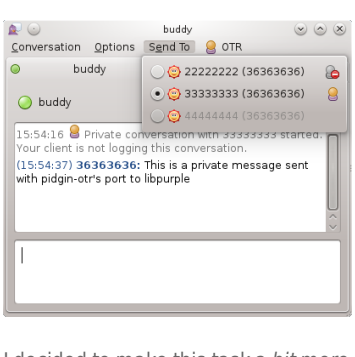
Encrypting the message by A seems to be the solution, but it creates another set of problems. To make user A able to encrypt a message for user B, he have to obtain a B's public key. But how to obtain it in a secure way? What, if B loose his key, used to secure a long history of conversations? Also, to make user B sure, that the A sent the message, user A have to sign it. But what, if user A says something humiliating to user B in private, but the second one reveals the message to the public and use it against A? The message is signed by A, so there is a proof he wrote it.

And there comes the Off-the-Record Messaging protocol. It's goal is to provide truly private conversations over any underlying IM protocol (such as xmpp, ICQ, Gadu-Gadu) in a easy to use manner. It ensures four aspects of privacy:

- **encryption**, so nobody other than A or B can read their messages;
- **authentication** of the other party, to make sure we are talking with the right person;
- **deniability**, because messages does not have digital signatures that are checkable by a third party – the other side of conversation cannot prove, the messages he got are sent by you (but he's still sure about it);
- **perfect forward secrecy** – if someone get your private keys, he won't be able to decrypt any *past* conversations.

OTR and Pidgin/libpurple

There is a plugin for Pidgin, implementing OTR messaging – pidgin-otr. Unfortunately, the user have to install it by himself, so it's not as easy to use as in other messengers. I was asked to integrate it within the official release.

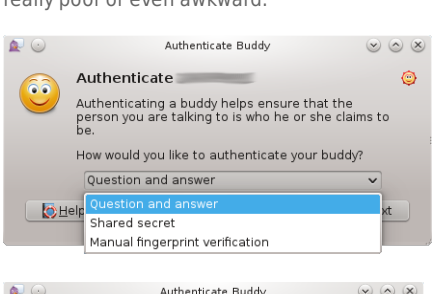


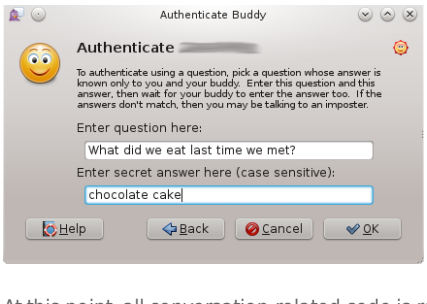
I decided to make this task a *bit* more challenging. Pidgin-otr is a Pidgin UI client plugin, so it's not available for other libpurple clients. Some of them (Adium) implements OTR by itself, some (Finch) doesn't offer such functionality. I decided to rewrite it as a pure libpurple plugin, to provide this excellent feature for all clients that use libpurple as a backend. Fortunately, pidgin-otr is pretty well designed, so I had only to alter things related directly to the UI.

There already was a similar attempt, named purple-otr. Its main problem was really poor UI – its author used pretty limited libpurple's Request API to create dialogs, so he wasn't able to clone all pidgin-otr's functionality. My situation is way more comfortable, because I am a Pidgin developer (having direct impact on its code), working on a 3.0.0 version (which breaks API, so I don't have to care about compatibility). That means, I was able to extend libpurple's features to better fit OTR plugin needs. *Some* of libpurple API changes that made it possible:

- **End-to-End encryption providers API** allows to present (implementation independently) conversation's security state. Previously, pidgin-otr and similiar plugins placed their own controls in various parts of the conversation window, now it's standardized.
- **Request API refactored with PurpleRequestCommonParameters**, which makes this API easily extendable.
- Using PurpleRequestCommonParameters for new features implementation, like an option to provide **HTML decorated text descriptions**, an option to **alter the dialog icon**, to make better control over the **dialog buttons**.
- Adding a new window type: **cancellable "please wait" dialog**, with an optional progress bar.

These changes may not look crucial. In fact, without them the libpurple's pidgin-otr port would look really poor or even awkward.





At this point, all conversation-related code is ready and working fine, and it's possible to use it conveniently. I haven't implemented configuration dialogs yet (It's the next task for me) nor adapted Finch for the new API (so it's not possible yet to use it with OTR, but it's not the hard part).

Unfortunately, there is no possibility to test it at the moment. The libpurple's part is in its repository, but the new pidgin-otr is developed in its non-public branch in OTR authors repo.

101 HTTP implementations

August 26, 2013 10:47 AM by Tomasz Wasilczyk

Previous libpurple version suffers from poor HTTP implementation. Ordinary user won't notice that, because plugins tries to fill the hole. However, when every single component that uses HTTP have to deal with the same issues, there must be some mess left.

Nearly a year ago, I've decided to put some effort here and implement new, flexible HTTP API. Now, I've came even deeper, replacing existing implementations with my new tool.

Replacing old HTTP API with the new one consisted of few stages. Firstly, I've replaced all `purple_util_fetch_url` occurrences with `purple_http_get`. That was the pretty easy step, because both functions does roughly the same thing: gets the URL and returns its contents. Then, it was the time for the tricky part: `purple_util_fetch_url_request` required building and parsing HTTP headers manually. Also, it leaves part of proxy handling on the caller in a weird way – he have to pass an full URL in request header, if proxy is on, or short (without hostname) otherwise.

Getting rid of `purple_util_fetch_url*` routines raised code quality pretty well, but the most difficult task was still undone. Some protocols had its own HTTP implementations – just imagine that amount of copy-paste. I'd like to share some examples:

- oscar (ICQ, AIM) implemented it on it's own to just perform simple GET request; the implementation was so horrible, that the author himself named the functions `straight_to_hell`, `damn_you` and `struct pieceofcrap`; fortunately, this code wasn't used for a long time;
- mxit: this could be implemented using old HTTP API, I have no idea, why author hadn't done that;
- msn: this one was somehow justified – there were no Keep-Alive connections in old API, so the author did it by himself to gain some performance; the bad thing is, he did it twice (for HTTP relays and SOAP handling);
- xmpp also has two implementations, just not that obviously superfluous: the BOSH implementation was complex and still full of TODO's; it took advantage of simple Keep-Alive request, but in really obfuscated way; rewriting of second implementation (out-of-band file transfers) was, by contrast, easy and pleasing task (especially, because I've done similar one within Gadu-Gadu protocol before);
- yahoo protocol plugin was the record-holder with the value of four distinct HTTP implementations; I think, only in case of yahoo file transfers it wouldn't be possible to do it using existing API.

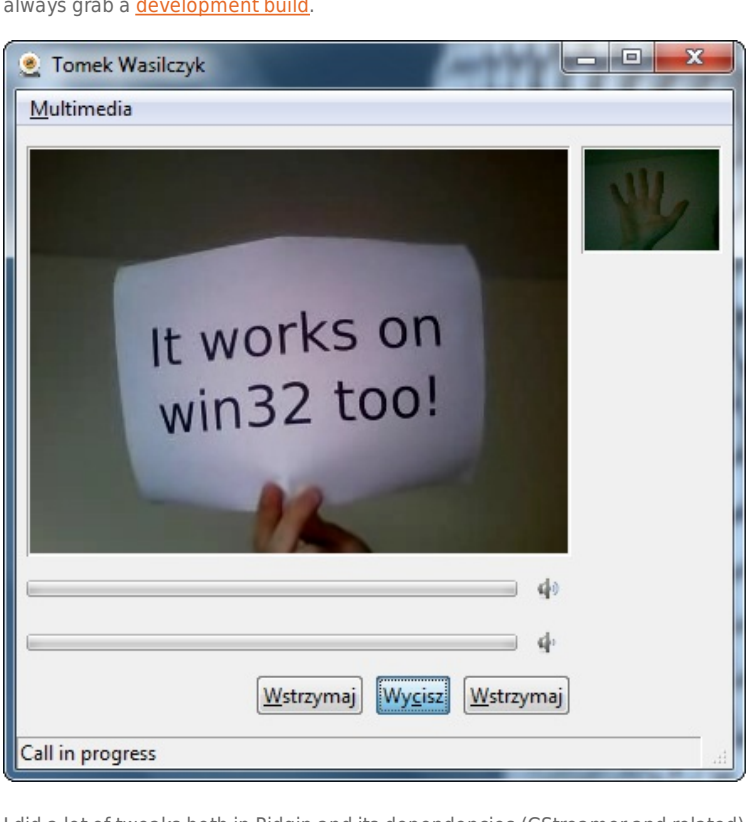
There are still two remaining, but I'm not sure if I will refactor them at all. The first one is for Gadu-Gadu protocol – it's included in libgadu library, so it's not trivial to pull it out without violating its API. The second is for Novell GroupWise Instant Messenger – a closed service, available for companies. There are no public test servers, which I could try out. Also installation and configuration of its demo overwhelmed me (in fact, it's buggy). I'm not sure if it's worth the effort to do both cases above.

My next task will be Off-the-record plugin refinement and integration into the main tree.

Pidgin with Video finally for Windows

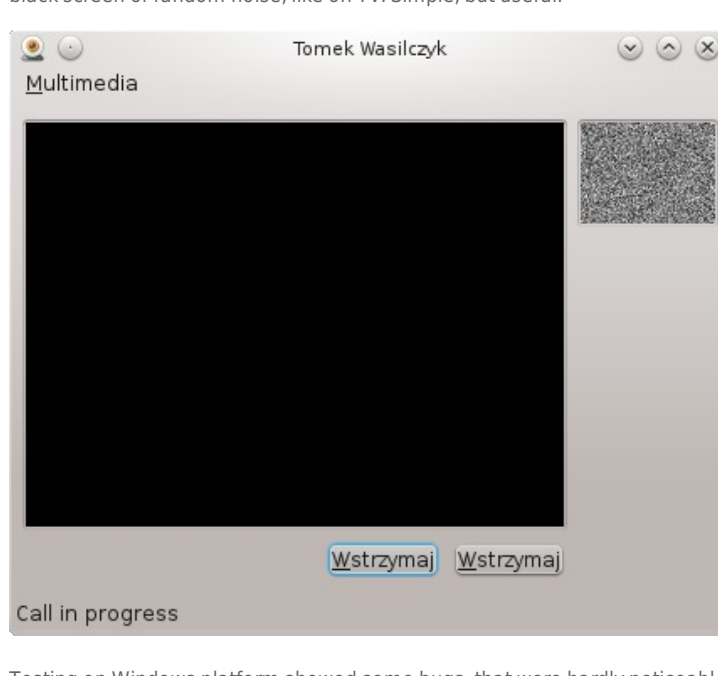
June 08, 2013 05:54 PM by Tomasz Wasilczyk

I just did another long desired feature: Video conferences support for Windows. There already were attempts to do it, but they were not finished nor even published – this one is already in Pidgin's tree, so it's a matter of time to get them released (*just* wait for 3.0.0). If you don't want to wait, you can always grab a development build.



I did a lot of tweaks both in Pidgin and its dependencies (GStreamer and related), but finally everything looks working and stable. The hard part, unlike in the previous Eion's attempt, was the camera capture plugins for GStreamer – DirectShow and WinKS. The first one is not buildable in newer gst-plugins-bad releases (at least, for mingw), the second is buggy. I've chosen to work with WinKS: the main problem was, it had broken support for different capture resolutions (I guess, old cameras had no support for it, so they were not affected). Many hours of debugging resulted in a simple patch, that makes the resolution fixed. The other one existed in Pidgin itself: there were no possibility to select camera, because winks used different method of device enumeration, than Pidgin supported.

On the occasion of testing VV on Windows, I've came up with a simple idea: some users may do not want to show their faces in video conferences, but they could want to see others. So, I implemented a new, virtual device (both for Linux and Windows): *Disabled*. Depending on user's choice, it displays black screen or random noise, like on TV. Simple, but useful.



Testing on Windows platform showed some bugs, that were hardly noticeable before: hangs on video testing, displaying a video output in a separate window. Beyond them, I also fixed some memleaks and did other tiny fixes.

The source code is available directly from the hg repository and openSUSE Build Service project (for dependencies). If you just want to test the VV feature on Windows, you can just grab the offline installer.

As always, I'm waiting for any feedback, bug reports and comments.

Four Pidgin Summer of Code students

May 28, 2013 12:52 AM by Mark Doliner

Pidgin was awarded four students for this year's Google Summer of Code. It was a difficult process to select just four students from the 34 great applications we received. These are the projects we finally chose:

- Ankit Vani will be working on GObjectification. This entails a lot of behind-the-scenes changes to the Pidgin code to use GObjects to make developer's lives easier.
- Ashish Gupta will be working on improving file transfer between libpurple and non-libpurple IM clients. He'll initially be focusing on the Yahoo! and XMPP protocols before moving on to a protocol-agnostic file transfer plugin.

- Bhaskar Kandiyal will be creating a website for browsing available Pidgin plugins, as well as improving Pidgin's plugin management and installation UI.
- Phil Hannent will create Quail – a Qt GUI for libpurple.

We're looking forward to seeing what they create! The coding period begins June 17 and ends September 23.

As always, thanks to everyone who applied. And remember, this is an open source project and you're welcome to contribute even if you're not participating in Summer of Code.

Students: Apply to Pidgin Google Summer of Code now!

April 25, 2013 05:31 AM by Mark Doliner

The application period for applying to Google Summer of Code opened on Monday and we've already received a number of applications. The application period closes next Friday, May 3rd. Just 8 days left—don't wait, submit your application soon!

Pidgin in Google Summer of Code 2013

April 09, 2013 07:21 AM by Mark Doliner

Google has accepted Pidgin into Google Summer of Code 2013. Woo-hoo! We're looking forward to mentoring a few lucky students again this year.

For more information, read Google's announcement, peruse our application template, and see our list of project ideas. The application period beings April 22nd—just two short weeks away!

We always encourage our users to brainstorm and share your ideas on improvements you would like to see in Pidgin, Finch, and libpurple. Feel free to share thoughts and ask questions on our devel mailing list.

Pidgin and the Impending Shutdown of Windows Live Messenger

November 11, 2012 11:58 PM by John Bailey

So, Microsoft recently announced that they'll terminate the Windows Live Messenger service in favor of Skype in early 2013. We've been getting a number of questions about what this means for Pidgin. Quite honestly, we don't know. At this point, all we know is that China will still be able to use Windows Live Messenger. That leads us to believe that the servers providing MSNP service will remain active and maintained for some period of time after the announced shutdown, but it's not clear whether or not that will be the case. It's also not clear if the servers supporting China's continued use of WLM will be accessible to non-Chinese IP space. Even further, it's not clear if the recently-launched XMPP interface to the WLM network will remain functional. We don't support that yet though, as it requires some authentication magic we don't implement. Even if we implement support for the authentication this XMPP gateway requires, it could end up being a waste of time, as it could get shut down at any time, either before or after the rest of WLM.

And before anyone goes there, we can't support Skype. There is no documentation of the protocol available to us, nor is there code we can borrow from a cleanly reverse-engineered alternative implementation. All that exists is SkypeKit, whose license agreement explicitly forbids its use in open-source software. The license also forbids use in "server applications" which precludes doing something like wrapping a simple closed-source XMPP daemon around SkypeKit. It is not currently possible to legally support Skype, so we won't try.

The bottom line is we have no idea what the announcement means for Pidgin or any other alternative clients yet. We'll all just have to wait and see.

Announcing our four Summer of Code students!

April 24, 2012 08:19 AM by Mark Doliner

We're pleased to announce that we've accepted four students for this year's Summer of Code!

- Gadu-Gadu PRPL improvements by Tomasz Wasilczyk, mentored by Ethan Blanton
- Plugin website by Nikhil Bafna, mentored by Kevin Stange
- Usage stats collection by Sanket Agarwal, mentored by Eion Robb
- libpurple on Android by Michael Zangl, mentored by Mark Doliner

It's always difficult to narrow down so many great applications into just a handful, and we want to thank everyone who applied. The coding period runs from May 21 through August 24. If you want to follow the progress of the four students, they'll be providing periodic status updates to our devel mailing list throughout the summer.

Read Google's official announcement here.

Libpurple in GSoC 2012

March 27, 2012 12:28 PM by Jorge Villaseñor

Libpurple was accepted in the Google Summer of Code this year 2012.

I urge every student reading this to apply for any of the projects accepted and if you like, apply to Libpurple.

We have a set of proposed ideas but you are encouraged to bring your own ideas since they will be fresher and will not compete with other people over the same project.

You can find libpurple's application page at Pidgin, Finch and libpurple.

Pidgin Accepted to 2012 Summer of Code!

March 18, 2012 02:11 AM by Mark Doliner

Good news, everyone! Google has accepted the Pidgin project's application to be a mentoring organization in this year's Google Summer of Code. If you love programming and are looking for a chance to help an open source project, look no further.

How much can you accomplish in a single summer? Quite a lot. To give you an idea, here's a list of some of our heftier projects of past years:

- SSL certificate verification and management
- Voice and video chat for XMPP
- The Bonjour protocol plugin
- The MySpace protocol plugin
- The SIMPLE protocol plugin
- Finch (command-line based IM client based on libpurple)

Details

- Get inspired by our ideas list. But don't limit yourself to those ideas—we love when students propose their own projects.
- The application period starts March 26 and ends April 6th (full timeline)
- Once the application period opens, apply here
- We're guessing we'll request slots for 3 students this year.
- If IM isn't your thing but you still want to participate, check out the list of other great organizations

Major Changes Afoot

August 28, 2011 03:44 PM by John Bailey

Well, it's been about forever since I last bothered to post anything here. Since my last post, we've released several times, introducing and fixing a bunch of bugs. Now, however, we're shifting our focus to new development of a sort that we don't do often—compatibility breakage and big internal changes. This means that our main development effort is now in working toward Pidgin and libpurple 3.0.0. I'm going to try to explain some of the work going on for the benefit of anyone who reads my ramblings, so here goes.

Versioning

First of all, there seems to be some confusion about how Pidgin and libpurple version numbers work. I'd like to try to clear some of that up.

Pidgin, Finch, and libpurple use what's called "Semantic Versioning." That is, each part of the version number has a particular meaning for users and developers. We chose this scheme to assist plugin developers in knowing when significant changes would require effort on their part to maintain compatibility with current Pidgin and/or libpurple versions. It also helps our users by letting them know when their existing plugins will stop working. So, how does this semantic versioning work? Let's look at the format of our version numbers and find out.

Pidgin, Finch, and libpurple version numbers have three components, separated by dots. At the time of this writing, 2.10.0 is the current version number. These components are called major (currently 2), minor (currently 10), and micro or patch (currently 0). Let's look at what each means.

- **Major**: the major version doesn't change often. The last time we changed it was in 2007 with the release of Pidgin 2.0.0, and before that was in 2004 with 1.0.0 when we started using semantic versioning under our previous name. Whenever this number changes, we've made changes to Pidgin, Finch, or libpurple that break compatibility with every UI and plugin that currently exists. Usually this means that we've removed something from the API exposed to plugins and UI's, or that we've changed something about a function (its name, arguments, return type, or the header file it's in). Sometimes plugins or UI's can just be recompiled when this happens; other times they need maintenance to become compatible with the new release. Additionally, the major version *never* decreases. It will *always* increase when it changes.
- **Minor**: the minor version changes more frequently than the major version, but generally less often than the micro or patch version. Whenever the minor number increases, we've added things to our API that do not break compatibility with existing plugins or UI's. A prime example of this is adding the voice and video support we added in 2.6.0. We added a bunch of stuff to the existing API, but didn't change anything that would cause a break in backward compatibility with existing plugins and UI's. When the minor version decreases (gets set back to 0), it means that the major version has changed, and the whole cycle starts over.
- **Micro** or **patch**: The patch version changes with almost every release. When this version increases, it means that we haven't touched API at all; instead we've done nothing but fix bugs or add small features that don't affect compatibility with plugins or UI's. When this version decreases (gets set back to 0), it means that the minor version has changed.

Related to all this, and important only for plugin and UI developers (so skip this paragraph if you're not a developer!), is the behavior of the `PURPLE_VERSION_CHECK` macro. Many developers expect `PURPLE_VERSION_CHECK(2, 5, 0)` or similar to expand to a statement that evaluates to 1 or TRUE in the case of building against libpurple 3.0.0. This, however, is *not* the case. Because our major versions are incompatible with each other, we have intentionally written `PURPLE_VERSION_CHECK` to fail if the major version is not an exact match. We understand this can be confusing and inconvenient, and we sincerely apologize for that, but we're not going to change it.

What all this means is that if the first number (major version) changes, you're going to need to update your plugins when you upgrade Pidgin, Finch, and libpurple. If the second (minor version) or third (micro version) numbers change, it means you need to upgrade Pidgin, Finch, and libpurple, but your existing plugins will still work. No matter what the Pidgin, Finch, and libpurple version numbers are, you should always be using the newest version.

Structure ("struct") Hiding

This paragraph is for those who aren't programmers. Feel free to skip it if you don't care about it. Pidgin is written in the C language. C has a type system, which means that if you declare a variable (imagine this as a box somewhere in your computer's memory) you can store only one type of data in it. A structure, or struct in C parlance, is a type made up of other types arranged sequentially. This is pretty easy to picture if you think of Lego blocks--stack a red block, a blue block, a green block, a yellow block, and a white block on top of each other and connect them together and you now have a structure made out of Lego blocks. It's pretty similar in C, except that you're telling the compiler to assemble something out of sequential boxes of memory instead of little plastic blocks.

Pidgin uses structs everywhere. We use them to represent things like your buddies, conversations, accounts, etc. Currently in Pidgin 2.10.0, most of the structs are in the public API—that is, anyone can directly access the members of the structs and do whatever they like. This is all fine and well, but it means that if our code changes such that a particular struct needs to grow significantly by the addition of new members, we can't always do that without breaking backward compatibility. (Yes, we included padding in a number of our structs, but we've burned through the padding in several of them, and although there are ways to work around it, I and a few other developers don't like them.) It also means that if we discover, for example, that switching the order of members in a struct allows the compiler to improve its optimizations or if we think a different order makes more logical sense for those of us reading and maintaining the code, we absolutely can't do this without breaking compatibility. We also can't rename members of structs for the same reason—it breaks compatibility with existing plugins, UI's, etc.

Because having these structs in the public API limits us so much, we're striving to hide as many of them as possible. By hiding, we mean that we're removing the struct definitions from the header files and moving them to the .c files, thus making them private. Plugin and UI authors will still be able to reference the "objects" with pointers, passing them to functions and operating on them with the appropriate sections of our public API, but no longer will the members of the structs be directly accessible outside of the .c files that define the functions that interact with them. For example, we have a PurpleRoomlist struct in libpurple/roomlist.c and libpurple/roomlist.h. For 3.0.0, the struct definition is in roomlist.c; thus the individual members are not directly accessible outside roomlist.c, even in other parts of libpurple. This "hiding" of the strcuts allows us significant internal flexibility in each file to modify the struct as we see fit.

Clean-Slate API Documentation

Because we generally change so much each time we increase our major version number, the API documentation can get a bit confusing if we keep doing `@since` tags and whatnot in our doxygen documentation. Our general overall feeling is that we prefer just starting with a clean slate at each new major version. This means that each time we do a new major version, all existing `@since` tags will disappear, any functions marked as `@deprecated` will be removed, renamed, or replaced as described in the `@deprecated` statement, and so on. We realize this decision may make some things more difficult for some plugin and UI authors, and we apologize for that, but our aim is to have overall cleaner documentation for everyone.

Merging of Old Projects

Over the years that we've participated in Google's Summer of Code program, we've accumulated a number of branches that have been sitting for quite some time. Most of these need some form of TLC and integration work. We want to try to incorporate at least some of them into 3.0.0 so we can finally benefit from the fruits of the students' labor. Most notably, we've been talking about merging the webkit integration branch into what will become 3.0.0. Eventually, this would allow the support of Adium's message styles, although it may not happen right away.

Another project of notable interest is some of the logging changes that went on in a previous Summer of Code project. One of our new Crazy Patch Writers took some of that work and made some progress on it; we don't know if this will make it for 3.0.0 or not yet, but it would be nice to have some of the features, such as non-blocking log writing.

Other Changes As Wanted

We may decide to make other changes since a major version change gives us the opportunity to break so much. There have been a number of ideas floated, ranging from supporting that XDG directory spec that I can't stand to doing away with the xml files in .purple and replacing them with something else (with what, in particular, has not seriously been discussed). There is a whole range of possibilities of things we could do for 3.0.0; it's just a matter of one of us wanting it and sitting down to write it.

So everyone should stay tuned, as we'll be making more and more changes over the course of the 3.0.0 development cycle. We don't yet know when we'll be releasing 3.0.0—this is another one of our famous "when it's ready, and not a minute before" time frames. We do know, however, that it will change a lot!

Current AIM Issues

February 23, 2011 08:50 PM by John Bailey

Over the last couple hours, we've had nearly everyone and his/her brother in #pidgin asking about connections to AIM causing a certificate prompt. The specific prompt is for bos.oscar.aol.com. The issue here appears to be that AOL has let the certificate expire. Because our certificate validation is more strict than some other applications, Pidgin users will get this prompt at every connection until AOL installs an updated certificate.

To resolve the prompt, you can make one of a couple choices. You can choose to trust that the connection is fine even though the certificate is expired and click Accept, or you can take the safe route and click Reject until AOL replaces the certificate.

To reiterate, this is not a Pidgin problem, but an AIM server problem.

Complex Transient Statuses for Quick Effect

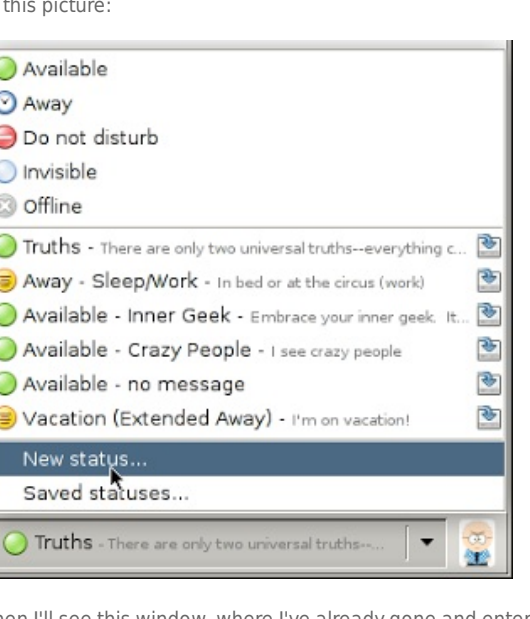
February 04, 2011 10:36 PM by John Bailey

In an earlier post, I discussed how to take advantage of Pidgin's "saved status" feature. After seeing some recent confusion in #pidgin about our status features, I decided it would be a good idea to give a quick overview on how to take advantage of a feature I never use--creating statuses (including complex statuses) from the status selector on the buddy list window.

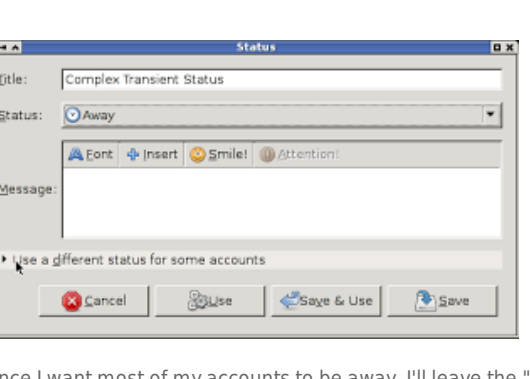
The status selector feels like it's been around forever. Prior to this, we had a rather horrible interface to "status" whereby you could either globally set all accounts to "away" or go to each individual account and configure a given status (away, do not disturb, vacation, etc.). All this was done via a single menu. For those of you who don't remember it, let me just say that it sucked. Someone (I think it was Sean Egan) threw that whole UI out the window and came up with the current status selector that reminds me a lot of the old Windows ICQ 99 client that had a pretty similar status selector. Ours is a bit more sophisticated, though.

The simple use of the status selector is fairly obvious--pick a status and type a message. In fact, this is what the majority of our users do when they change statuses. This is what is called a transient status. What most people don't know, however, is that you can create complex statuses, such as having your MSN account set to "Busy" while your AIM account is set to "Invisible." Let's give a quick example.

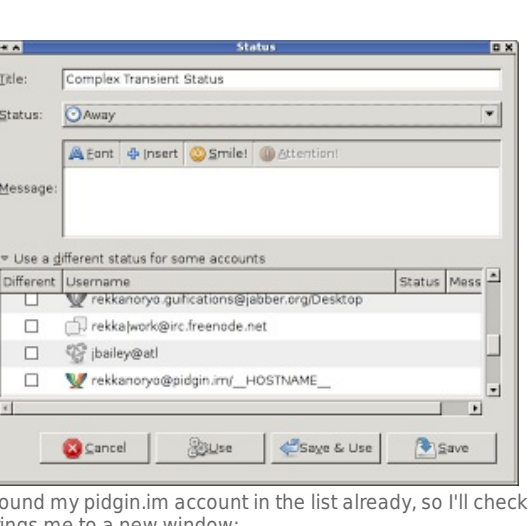
I have a number of accounts in Pidgin. Let's say I want to have my pidgin.im XMPP account set to "Available" but the rest of my accounts set to "Away" to create a simplistic scenario that's really easy for me to snag a screenshot of. To do this, I'll go to the status selector and select "New status..." like in this picture:



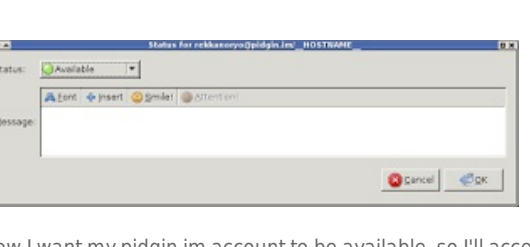
Then I'll see this window, where I've already gone and entered a title for the status. This is the name you'll see in the status selector and in the middle section of the status selector's menu. If I wanted a message to go with the away status, I'd enter it here now, as well.



Since I want most of my accounts to be away, I'll leave the "Status" selection as "Away." As you can see, I have my mouse pointer over "Use a different status for some accounts," which when expanded will allow me to set statuses for individual accounts, like so:



I found my pidgin.im account in the list already, so I'll check the box in the "Different" collumn. That brings me to a new window:



Now I want my pidgin.im account to be available, so I'll accept what I see. If I wanted a message here, I'd type it in the "Message:" box. When I'm happy with what I have, I'll click OK. Notice that the "Status" column in the previous window changes to reflect what you selected for the individual account.

Since I'm happy with what I have, I'll now click "Use", which applies the status. If I wanted to make this a saved status that is saved permanently, I could click the "Save" button or the "Save & Use" button if I wanted to also immediately apply it when saving.

That's all there is to it!

Fun at the GSoC Mentors Summit

November 06, 2010 03:21 AM by Elliott Sales de Andrade

Last weekend was the GSoC Mentors Summit. As a mentor for the Pidgin, Finch and libpurple project, I attended for the first time this year.

It was pretty interesting and a lot of fun, but I have to say I didn't really feel like much of a geek there! Everyone's either got an iPhone or a Droid, and they've all done awesome stuff. Can you say you've worked on WordPress, used by millions of websites, or Apache, serving even more websites, or RTEMS, running several space exploration instruments and other consumer products, or who knows what else?

Nevertheless, it wasn't like I was totally unknown. There were quite a few people who use Pidgin, even one or two using Finch. Oh, and some Mac users using Adium (using libpurple), too.

The flight was fine, but the weather was a bit disappointed. It would have been nicer to explore a bit more, but at least I got to eat a ton of chocolate. I'll try and write a longer re-cap later.

MSNP16 and SLP-rewrite merged

November 01, 2010 01:01 AM by Jorge Villaseñor

I have just pushed the revision that merges my MSNP16 and SLP branches to the main development branch in pidgin. I'm very happy to have this branches merged since they represent almost all the code I have been writing on the last year.

Yes I have started coding MSNP16 support almost a year ago and it took a lot of effort, reverse engineering, debugging Wireshark dumps and a lot of pidgin debug logs to get it working. That is a lot of time!

It is true that the MSNP16 code was almost complete when I started my SoC work but I though it would be better to start the SLP rewrite over the MSNP16 branch to be able to easily test both codes at the same time and try to get it in a better shape before merging it to i.p.p.

I know I have announced this merge like two weeks ago, but you know, I wanted this merge to be followed by a reasonable "beta" testing before being released and at that time it got that we had an security issue and needed to release 2.7.4. Once it was out, there were some ICQ issues that needed a quick release to fix that bugs, so we got a 2.7.5. Now I was able to merge and get a normal release cycle to get beta testers to find bugs in this new and nice code.

I hope this code will fix more issues than it brings up, specially the ones related to data transfer. Since most of the code on this area have changed due DirectConn and SLP-rewrite, I guess it would be a good idea to review and close most of the related tickets since the traceback and debug output would be really useless now. Yei for smashing tickets!

I hope you all like 2.7.6 when it get released!

Death of a thousand tickets

October 26, 2010 02:06 AM by John Bailey

Well, by now it's obvious to the world that I kicked Pidgin 2.7.4 out of the nest last week. Although that release included some nice new features for ICQ users, lots of bug fixes, and some remote crash fixes, it's not without its share of problems. Those problems are producing a bunch of duplicate tickets for us to deal with, so I thought it might be a good idea to post about them here just in case anyone bothers to read my rambling.

The first problem is the AIM/ICQ chat bug. When using a multi-user chat on AIM or ICQ, no messages can be sent. You'll get an error stating your message was too long. This was an unintended side effect of merging the work of one of our Summer of Code students. Ivan accidentally removed a line of code that he shouldn't have. Yes, you read that right--a single line of code. Apparently it was pretty important! Ivan restored that line and things will be working as they should in 2.7.5 when we release it.

The second problem is an old one that's come back with a vengeance. This one is an ICQ issue where messages that contain
disappear because they're treated as HTML tags. This seems to be specific to combinations of Pidgin and other non-official clients. Pidgin recently gained some new ICQ protocol-level features related to formatted messages; some non-official clients, including older versions of Pidgin, don't handle it gracefully. At this point, we've done all we can do about it; the other clients will have to step up and make fixes to handle the messages correctly now.

The next problem is another wonderful ICQ encoding bug. I'm not sure if this one is a side effect of Ivan's work or not, but Ivan did do a lot of work on encoding problems on ICQ. We all thought things were improving, but apparently there are still a few odd cases (and a crapton of stupid ICQ clients that should be purged from existence) in which our handling of encodings just isn't quite right. We're aiming to have this fixed for 2.7.5, but it's not quite done yet.

Is it obvious I wish ICQ would just disappear immediately and permanently?

The last "big" problem that seems to be cropping up is a crash related to MSN file transfers. When using direct-connect file transfers, somewhere along the line we do something stupid internally that causes a crash. We thought we had it fixed for 2.7.4, but alas it still exists. We're looking into it and hope to get it fixed soon. In the meantime, it's pretty easy to prevent the crash--edit your MSN account and turn off the direct-connect file transfers. It's an option on the advanced tab of the account editor.

At any rate, since we know about these bugs already, please, *please*, ***please DON'T*** open new tickets about them!

Last updated: June 29, 2015 01:00 PM (*All times are UTC.*)

Powered by: PLANET